НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
“___”_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп’ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Веб додаток інформаційної системи тайм-трекінгу»

Виконав:
студент _IV_курсу, групи__ІП-64__
Царук Володимир Вікторович _____
(підпис)

Керівник:
Асистент
Аленін Олег Ігорович _____
(підпис)

Консультант з нормоконтролю:
Професор, доктор технічних наук
Симоненко Валерій Павлович _____
(підпис)

Рецензент

_____
(підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
"___"_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Царуку Володимиру Вікторовичу

1. Тема проєкту «Веб додаток інформаційної системи тайм-трекінгу»

керівник проєкту Аленін Олег Ігорович, асистент, затверджені наказом по університету від *« 07 » травня* 2020р. № 1081-с

2. Термін здачі студентом закінченого роботи 2020р.

3. Вихідні дані до проєкту *технічне завдання, теоретичні дані.*

4. Зміст пояснювальної записки: *огляд та аналіз існуючих рішень на ринку, вибір технологій для розробки та підтримки даної системи, проектування системи тайм-трекінгу, проектування сховища даних, впровадження бізнес-логіки системи, демонстрація роботоспроможності додатку.*

5. Консультант роботи, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Симоненко В.П.		

6. Дата видачі завдання *01.09 .2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1.	*Затвердження теми роботи*	01.08.2019	
2.	*Вивчення та аналіз завдання*	15.09.2019-15.03.2020	
3.	*Розробка архітектури та загальної структури систем*	15.03.2020-25.03.2020	
4.	*Розробка структур окремих підсистем*	25.03.2020-05.04.2020	
5.	*Програмна реалізація системи*	05.04.2020-15.04.2020	
6.	*Оформлення пояснювальної записки*	15.04.2020-20.05.2020	
7.	*Передзахист*	26.05.2020	
8.	*Захист*	25.06.2020	

Студент Володимир ЦАРУК _____

(підпис)

Керівник Олег АЛЕНІН _____

(підпис)

Анотація

У бакалаврській роботі розглянута проблема коректного керування проектами, аналіз витраченого часу на виконання завдань.

Був запропонований та розроблений веб-додаток з можливістю оптимального керування персоналом на проектах та можливість фіксування затраченого часу на виконання завдань.

Реалізоване програмне забезпечення, яке дозволяє оптимальне керування проектом, а саме: реєстрація, створення нового проекту, створення нових завдань до проектів. У програмі присутні ролі, такі як: адмін, менеджер та розробник, які необхідні для взаємодії над проектом. Менеджер має можливість додавати розробників в проект, а також закріпляти за розробником завдання. В свою чергу розробник може вести трекінг затраченого часу на виконання завдання. Для створення програмного продукту було використано мову програмування Java з використанням середовища розробки Intellij IDEA 2019.3.1.

Annotation

The bachelor thesis deals with the problem of correct project management, analysis of time spent on tasks.

A web application has been proposed and developed with the ability to optimally manage project staff and capture time spent on tasks.

Implemented software that allows optimal project management, namely: registration, creation of a new project, creation of new tasks for projects. The program includes roles such as admin, manager and developer that are required to interact with the project. The manager has the ability to add developers to the project, as well as assign the developer to the task. In turn, the developer can keep track of the time spent on the task. Java programming language was used to create the software using Intellij IDEA 2019.3.1 development environment.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІАЛЦ.467100.001 ВП	Відомість проекту	1	
3	А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	70	
5	А4	ІАЛЦ.467100.004 Д1	Алгоритм користувацьких дій	1	
6	А3	ІАЛЦ.467100.005 Д2	Функціональна схема	1	
7	А3	ІАЛЦ.467100.006 Д3	Структурна схема	1	
8	А3	ІАЛЦ.467100.007 Д4	Текст програми	10	

					ІАЛЦ.467100.001 ВП
Зм.	*Арк.*	*№ документна*	*Підпис*	*Дата*	
Розробив		*Царук В.В.*			*Веб додаток інформаційної системи тайм-трекінгу Відомість дипломного проєкту*
Перевірив		*Аленін О. І.*			
Н. Контр.		*Сімоненко В. П.*			
Затверд.					

Літ.	*Аркуш*	*Акрушів*
	5	*1*

НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, гр. ІП-64

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня бакалавр

на тему: “ Веб-додаток інформаційної системи тайм-трекінгу ”

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання поширюється на розробку та подальшу підтримку програмного продукту на тему: ”Веб-додаток інформаційної системи тайм трекінгу”.

Область застосування – спрощення взаємодії з персоналом під час розробки програмного забезпечення, ведення та аналіз затраченого часу на виконані завдання.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даного програмного забезпечення виступає завдання для виконання роботи кваліфікаційно-освітнього рівня “бакалавр програмної інженерії”. Воно затверджено кафедрою обчислювальної техніки Національного технічного університету України “Київський політехнічний інститут ім. Ігоря Сікорського”.

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту являється дослідження та розробка програмного забезпечення в форматі веб-додатку, для спрощення керування роботою над проектами, а також ведення статистики затраченого часу.

Розробка призначена для компаній, які хочуть вдосконалити свою систему управління та для персонального аналізу затраченого часу на виконані завдання.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерела, які були використані для написання являється науково-технічна література з створення веб-додатків на мові Java та різні публікації в Інтернеті.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробляємого продукту

Вимоги, які поставлені до даного програмного забезпечення є такими:

- надати простий та інтуїтивно зрозумілий інтерфейс, а також необхідні функціонал для ефективного управління персоналом;
- бути уніфікованим для різної локалізації в залежності від вибраної мови;
- додавання нових та видалення старих проектів та завдань;
- пошук по назві проекту чи завдання;
- виставлення затраченого часу на завдання;
- додавання відповідальної команди за проект чи за завдання.

5.2. Вимоги до програмного забезпечення

- Операційна система Windows або Linux/Unix
- Java 8+
- Intellij IDEA 2019.3.1
- Node-v8.17.0
- PostgreSQL v12

5.3. Вимоги до апаратної частини

- Процесор рівня Intel Core i3 і вище.
- Вільне місце на жорсткому диску не менше 128 ГБ.
- Оперативна пам’ять не менше 4 ГБ.

6. ЕТАПИ РОЗРОБКИ

Етапи	Термін
Вивчення літератури	22.12.2019
Складання та узгодження технічного завдання	22.12.2019
Оглад та аналіз ресурсів для розробки	15.01.2020
Проектування бази даних	20.01.2020
Розробка програмного продукту	11.02.2020
Тестування розробленої частини	01.05.2020
Налагодження та виправлення помилок	15.05.2020
Оформлення документації дипломного проекту	06.06.2020

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проєкту
на тему: «Веб додаток інформаційної системи тайм-трекінгу»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTML – HyperText Markup Language, мова, яка була створена і використовується, для розмітки документів в мережі.

JSX – повна назва JavaScript XML використовується для розширення синтаксису, використання даного розширення надає можливість використовувати синтаксис схожий на HTML з метою опису структури інтерфейсу.

JIT − Just-in-time compilation, дана технологія розроблена для підвищення продуктивності в системах, що використовують байт-код, для цього тільки під час роботи самої програми відбувається трансляція самого байт-коду в машинний.

MVC – повна назва Model-View-Controller, це є архітектурний шаблон, який використовують для проектування веб-додатків.

БД – база даних.

ЕОМ – електронно-обчислювальна машина, тобто пристрій для обробки певної інформації.

HTTP − з англіської Hypertext Transfer Protocol, це протокол передачі гіпертекстових документів, часто з цією абревіатурою можна зустрітись при написанні веб-додатків.

СУБД – це система управління базами даних.

ACID − це набір властивостей, який притаманний транзакціям в базах даних і гарантує їх надійну роботу.

ІТ – інформаційні технології.

SQL − або Structured query language, це структурована мова запитів до баз даних.

SPA − або Single Page Application, динамічний веб-додаток, який як оболонку для всіх веб-сторінок, щоб взаємодіяти з користувачем, буде використовувати єдиний HTML-документ.

ПЗ – програмне забезпечення.

NoSQL− no only SQL, це такі база даних, в яких присутній інший механізм зберігання та взаємодії з даними, ніж такий підхід, що використовується в реляційних базах даних.

Front end − це користувацький інтерфейс, який використовують для взаємодії між користувачем і back end

JSON − або JavaScript Object Notation, це текстовий формат призначений для обміну даними.

Back end − це серверна частина програмного продукту, яка за певною бізнес-логікою реагує на запити від front end.

ВСТУП

З року в рік автоматизація, комп'ютерні системи стають невід'ємною частиною нашого життя. Важко уявити, що за останні десять років мережа Інтернет сягнула неймовірних темпів: майже більше половини населення планети використовує Інтернет в повсякденному житті і це тільки початок. Але все починалось з ідеї і як зараз говорять – все починалось з стартапу, проекту.

Щоб створити реально необхідну систему потрібно правильне та якісне керування. Саме поняття управління проектами почалося з виникненням людської цивілізації. Кожен день новітні технології змінюють наш світ. Починаючи від повсякденного приготування їжі, використовуючи мультиварку й завершуючи науково-технічним прогресом. Люди крок за кроком планують, впроваджують, розробляють та контролюють продукт.

Розберемо більш детально. Управління проектами – це організоване управління всього, що необхідно для досягнення мети. Саме тут необхідно враховувати два поняття: вчасно і в рамках бюджету, нехай то проведення маркетингової компанії, розробка програмного забезпечення, чи створення космічного шатлу – все це проектне управління, яке дозволяє домогтися успіху в поставленій цілі.

Із цього слідує, що однією із найважливіших характеристик ефективності роботи є швидкість передачі інформації і створення тісної співпраці в колективі.

Всі ідеї з яких починаються проекти дуже різні. Не існує ідеальної системи управління проектами, що підходить для кожного. Також не існує систем, які б чітко виконували всі ідеали керівника і були б зручними для всіх членів команди. Однак за весь час існування та вдосконалення проектного управління було створено багато ефективних підходів.

Мабуть всі розуміють, якщо людина сидить на робочому місці з восьомої години ранку до шостої години вечора це ще не означає, що вона працює ефективно і вона може охопити всі свої завдання. Працювати продуктивно – це означає, що ти використовуєш всі наявні ресурси по максимуму, з метою швидко і якісно вирішити поставлені завдання по проекту. Це питання можна вирішити за допомогою використання тайм-трекерів робочого часу, які допоможуть ефективно керувати проектом і вести облік часу для персонального аналізу. Також окрім основних функцій для підтримки та управління проектами, особливо для малих та середніх підприємств є дуже важливий факт як ціна продукту.

У подібних системах важливу роль відіграє керівник або як його ще називають менеджер по координуванню процесу. Він виступає ініціатором процесу і проводить контроль над виконаними завданням при цьому, якщо все буде коректно зроблено, а саме визначені основні ролі на проекті, скорість проектування і підтримки проекта буде в багато разів більша.

Ціллю даної бакалаврської роботи розробка саме такого продукту, яке дозволить ефективно керувати проектами, розробники з легкістю можуть відмічати свій час для розуміння загальної картини своїх знань, аналізуючи затрачений час. Такий продукт допоможе ефективно проводити не тільки робочий час, а й час відпочинку.

Лікарі рекомендують не брати завдання додому, потрібно організовувати виконання роботи таким чином, щоб вона займала найменше часу. Потрібно ретельно планувати час і слідкувати за ним. Також люди часто забувають про емоційне і фізичне перемикання, адже люди часто стикаються з проблемою емоційного вигорання. В іноземних компаніях вже давно працює правило, якщо робітник залишається на роботі довше ніж це прописано в трудовому договорі або робітник виконує завдання, яке б він мав зробити в офісі, а працює над ним вдома – це все може бути причиною його майбутнього звільнення. Процес керування передбачає чітку

послідовність організованих кроків, що полягає не тільки у плануванні завдань, але й в організованому слідуванні та постійному аналізу своїх дій.

Для того, щоб вирішити дану проблематику, яка забезпечить досягнення поставленої мети, першим кроком вирішимо наступні завдання:

- Огляд та аналіз готових рішень в сфері тайм-менеджменту або як його ще називають тайм-трекінгу
- Спроектуємо нову систему включаючи: БД, макет бачення проекту
- Розробка спроектованої backend частини системи використовуючи Java Spring Framework
- Розробка спроектованої frontend частини системи використовуючи AngularJS фреймворк
- Тестування створеного програмного забезпечення
- Створення інструкції по використанню продукту

РОЗДІЛ 1
ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ В СФЕРІ ТАЙМ-МЕНЕДЖМЕНТУ

1.1. Опис завдання

Завдання на дану роботу полягає в створенні програмного забезпечення в форматі веб-застосунку для релізації можливості ефективного управління проектами, ефективність проявляється не тільки в автоматизації створення проектів, завдань до проектів, а й можливість додавати відповідальних за завдання, які можуть вести статистику затраченого часу на виконанні завдання.

Кожне підприємство зможе використовувати цей додаток для коректої нормалізації роботи в сфері менеджементу. Працівники, які будуть вести статистику виконання, зможуть злегкістю аналізувати в на що в них пішло більше часу, які питання їм потрібно розібрати більш детально, щоб в подальшому не виникало проблем.

Система повинна бути розділена як на клієнтську так і на серверну частину. Все це дасть можливість зберігти данні, навіть якщо було втрачено пристірій, на якому була запущена і використовувалась клієнтська частина, для цього необхідно виконати повторну авторизацію на новому пристрої.

1.2. Аналіз існуючих рішень, які присутні на ринку

На сьогодішній день існують вже готові рішення, які довзолять спростити і покращити управління проектами і в цілому спростити життя людей. Кожен з цих розроблених застосунків відрізняється одне від одного, в них є свій унікальний функціонал, який вирішує спецефічні проблеми ринку.

Наші реалії говорять про те, що доступність ЕОМ сягнула на той рівень, що майже кожна людина має змогу користуватись подібними додатками, які автоматизують твій процес і надають можливість насолоджуватись процесом, не вдаваючись в рутинні проблеми. Під поняттям ЕОМ йдеться мова не тільки про ПК чи ноутбуки, до цього можемо

віднести й одноплатні комп'ютери, які на ринку крепко закріпились і в певних моментах стабільності та надійності не програють сучасним ПК.

Також телефони стали не від'ємною частиною нашого життя і важко уявити, щоб все це вже стало для нас буденністю. Кожен з цих девайсів, має доступ до інтернету, що дає можливість доступ одної і тої ж інформації з різних куточків нашої планети.

Бізнес в сфері програмного забезпечення завжди орієнтується на кінцевого користувача, на ту аудиторію, якій був присвячений продукт, це все дає можливість розвивати і продвигати свій продукт, орієнтувавши його не різноманітні платформи.

В наступному пункті буде розглянено програмні рішення, які вже використовуються на ринку, для подальшого аналізу.

Jira це продукт, який був представлений на ринку, як система відслідковування та нотування помилок, він приносить велику користь, організувавши ефективну взаємодії між розробниками, хоча на сьогоднішній день він використовується для управління проектами.

1.2.1. Jira Software

Jira є не безкоштовним продуктом і для її придбання є можливість зробити оплату в трьох видах. Існують такі види оплати: одноразовий і щомісячний платіж, а також річна підписка.

Ця система входить в трійку найкращих світових рішень з проблемою управління проектами. Часто цю систему можна зустріти в компаніях зв'язаних з сферою IT, які займаються розробкою програмного продукту. За допомогою Jira зручно створювати, планувати і вести контроль завданнь. Розробник даного продукту спешу створив робочу дошку, яка є головним елементом при запуску ПЗ. Головний екран даної системи зображений на рис. 1.1. Весь життєвий цикл розробки ПЗ можливо легко відстежити просто зайшовши в Jira і аналізувати роботу команди [1].

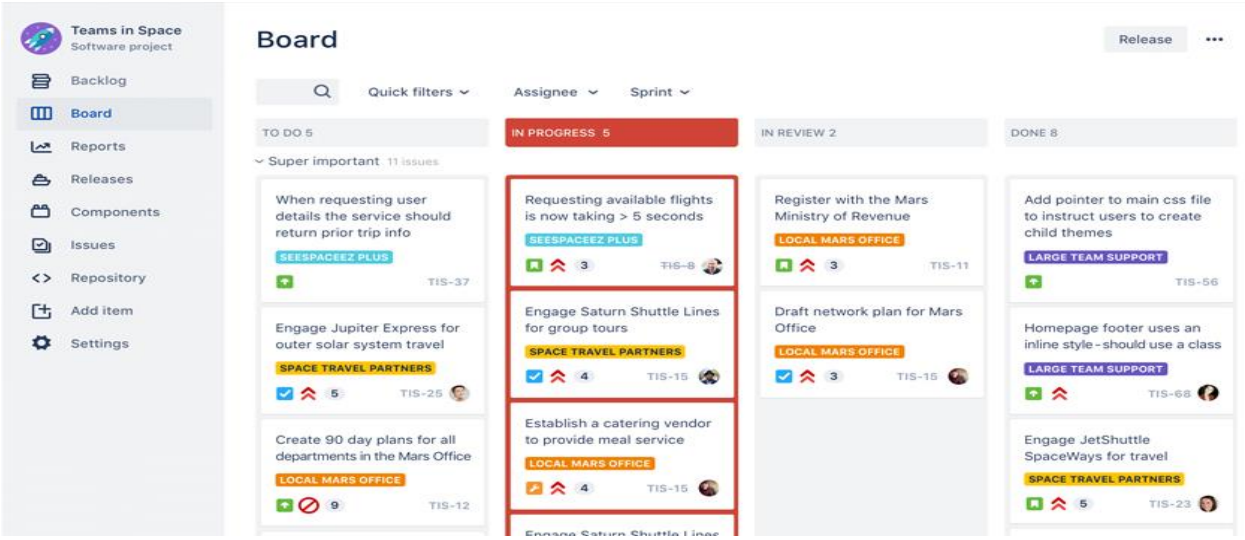


Рис 1.1. Головний екран Jira

Головною його ідеєю та бажанням було додати велику кількість різноманітних інструментів, за допомогою яких можна налаштувати все для ефективного рішення проблем. Jira можна використовувати в багатьох ситуаціях. Перше, для загального розуміння управління проектами, тобто маючи назву проекту, завдання, статуси виконань і подібні речі. Друге, для відстеження всіх процесів розробки, на якому етапі знаходиться виконане завдання, як його оцінили працівники. Приклад такого зображений на рис.1.2

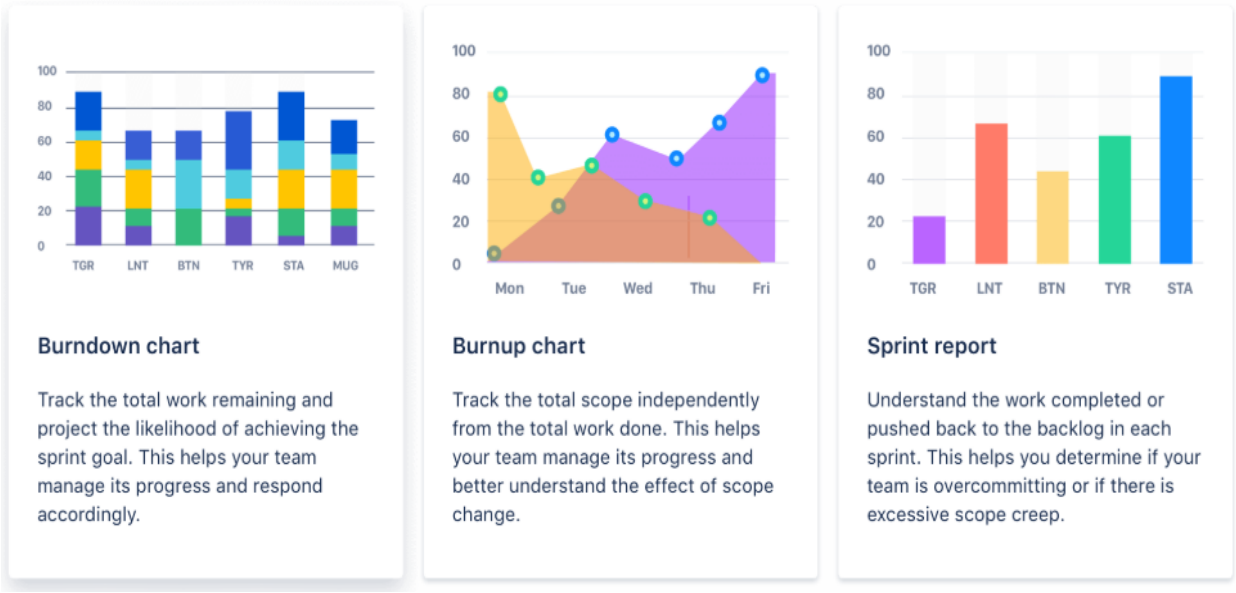


Рис 1.2. Інструменти для аналізу роботи в системі Jira

Так як ця система надає можливості змінювати свої налаштування її можна використовувати не тільки в сфері IT, наприклад для менеджменту персоналу економічної сфери. Jira надає підтримку локалізації таких мов як:

англійської, української, французької та інших мов. Дане ПЗ відмінно підходить для таких користувачів, які полюбляють технічно складне програмне забезпечення. Дана система є достатньо неефективною в специфічних питаннях і часто буває надмірно складною. Команди розробникі, персонал, який користуються кожен день даною системою спробували оптимізувати систему під себе, а саме змінити параметри налаштування, вони вважають, що цей процес оптимізації управління є занадто громіздким і важким для звичайного користувача [2].

В цій системі головною ідеєю є сам процес, процеси в системі Jira в свою чергу – це набір різноманітних статусів і переходів, через яке проходить помилка, задача, певна проблема протягом її життєвого цикла. Фактично, це демонструє внутрішні процеси вашої організації, його зображено на рис. 1.3.

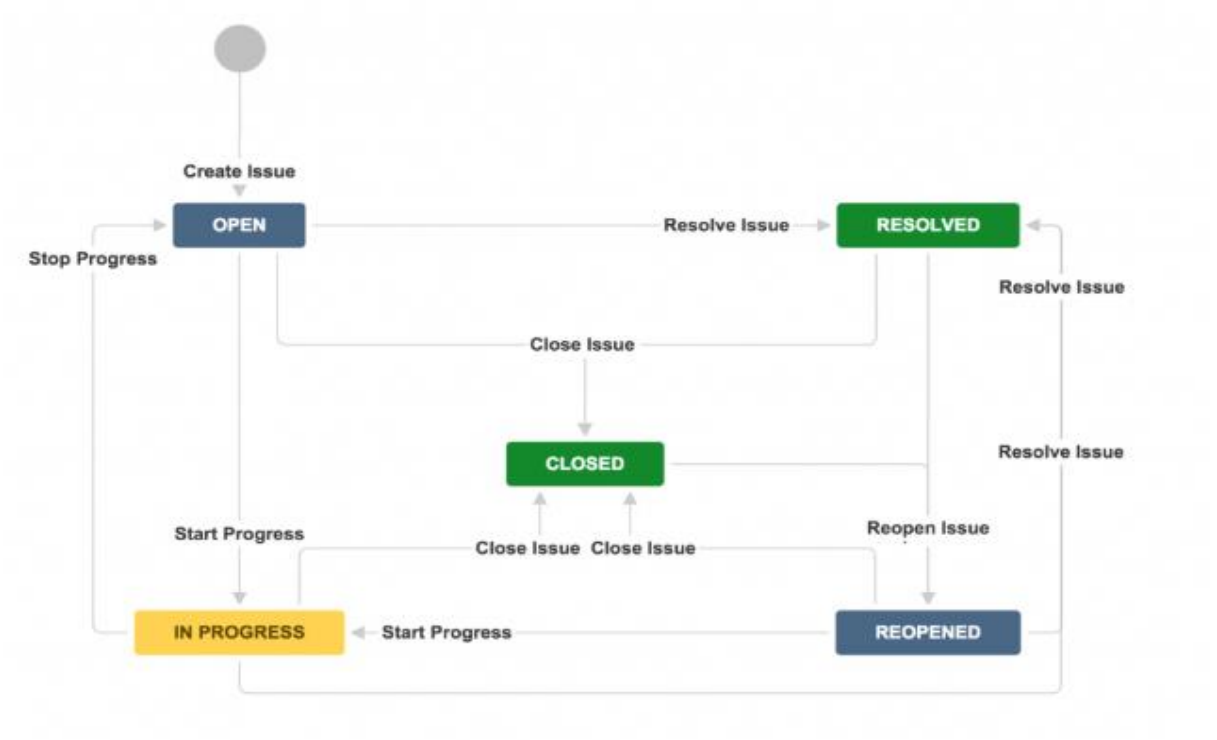


Рис 1.3. Типи процесів в Jira

В меню “Типи задач” є можливість переглянути всі типи задач, які дуже легко створити і відстежувати. Такі питання в JIRA класифікуються під різними формами, такими як нова функція, підзадача, помилка тощо, як зображено на рис. 1.4.

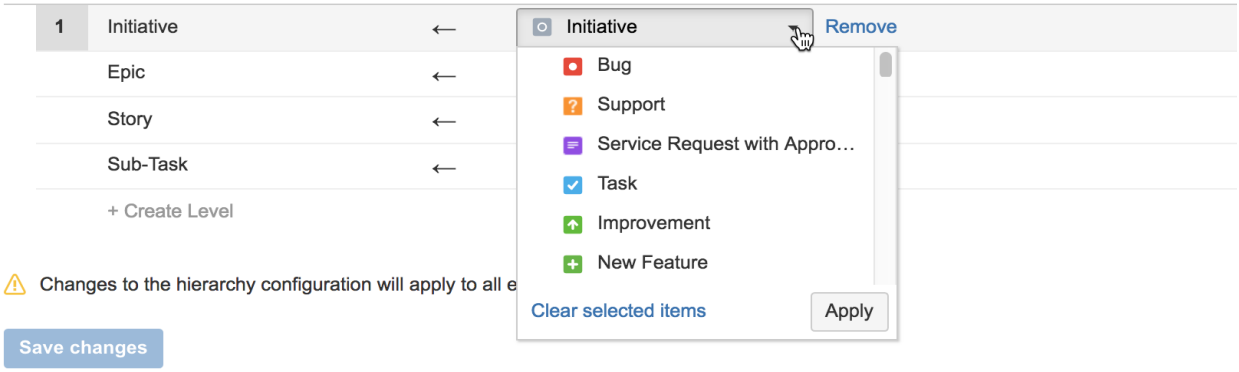


Рис 1.4. Типи завдань в Jira

В Jira існує дві системи організації типів завдань, як зображено на рис. 1.5. Перше – це стандартна система організації типів завдань її повна назва “Default Issue Type Scheme”. У стандартній системі організації типів завдань всі нові створені завдання автоматично додаються на схему. Agile Scrum система організації типів завдань її повна назва “Agile Scrum Issue Type Scheme”. В цій системі завдання і проекти, які асоціюються з Agile Scrum, використовують цю систему, окрім цих двох систем можна створювати і свої типи задач , просто налаштувавши функціонал під себе.

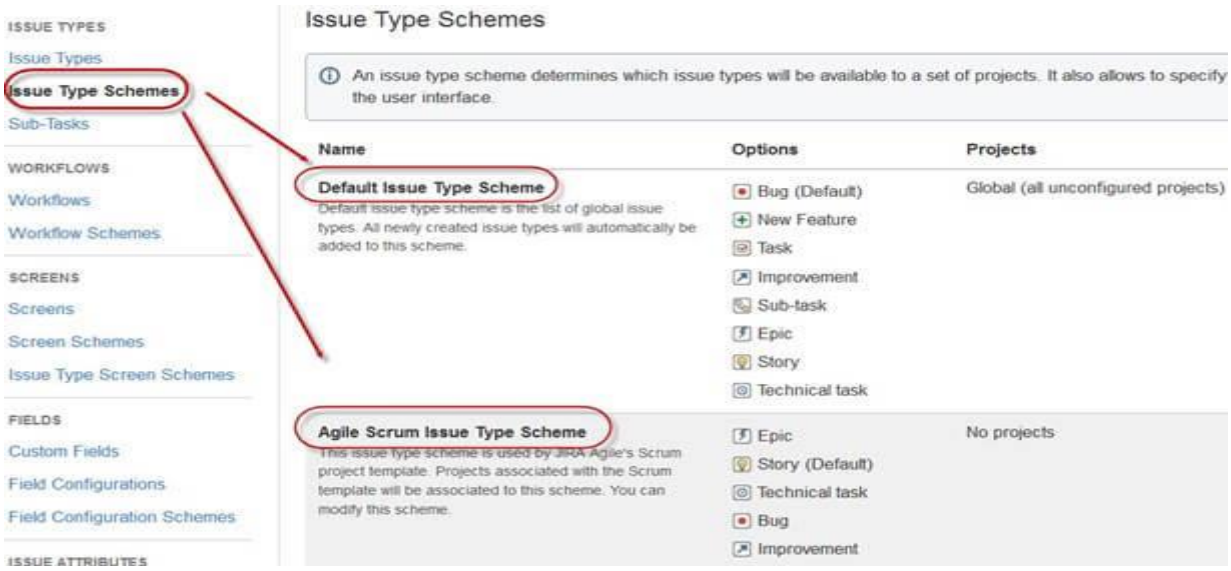


Рис 1.5. Організації типів завдань в Jira

Розглянемо основні можливості системи Jira[3]:

- Створення проектів і задач, які стосуються даного проекта
- Можливість пошуку по задачам в проекті
- Різноманітність різних фільтрів

- Можливість побудов різних статистичних діаграм для коректоного аналізу роботи команди
- Інтерактивність
- Зручне керування персоналом
- Наявність історій з можливістю підключення до таких систем управління версій як Bitbucket чи GitHub

1.2.2. Trello

Trello це сервіс, який є одним з найпростіших для початку роботи з системами управління проектів. Він є дуже гнучким та універсальним, для розуміння. Головна ідея даного інструменту це управління проектами в різних сферах таких як: комерційній, творчій, соціальній. Потрібно відмітити, що даний сервіс зручний для особистого планування і самоконтролю.

Також до позитивних сторін можна віднести такі речі як: можливість відразу спостерігати статуси різних завдань на головному екрані, зручна інтеграція по використовуванню з іншими популярними інструментами, які так необхідні для ефективної взаємодії.

Дану систему, через її продуктивний функціонал та високу гнучкості, можна використовувати як звичайний менеджер задач для нашого повсякденного використання. Зразок інтерфейсу даної системи зображений на рис. 1.6. При використанні Trello відразу можна відмітити візуальне оформлення, його інтерфейс збудований на основі канбан-дошок а саме: дошки з певною тематикою головного екрану і картки, які і виступають задачами [5].

Якщо говорити за ідею канбан-дошок, то для відразу потрібно пояснити, що це головний простір, в якому здійснюється майже вся частина роботи. На дошці є вертикальні списки їх називають листи, на яких знаходяться картки. Керування даними картками може відбуватись як того

забажає користувач, робити такі дії як: розміщати на будь-якій з дошок, перемістити в потрібне місце, копіювати тощо.

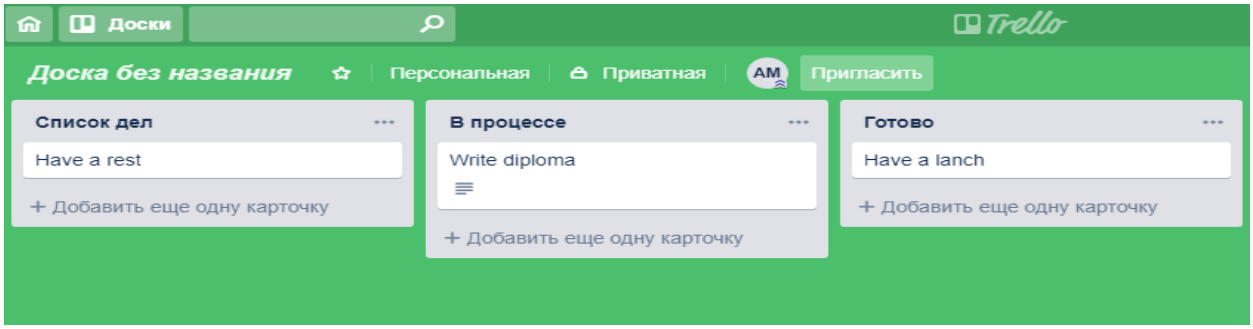


Рис 1.6. Зразок інтерфейсу сервісу Trello

Trello забезпечує можливість звужувати або навпаки розширювати зону бачення доски, це зображено на рис. 1.7. Доступні такі типи як:

1. Приватний – він надає можливість переглядати і змінювати вміст тільки тим учасникам, які додаються вибірково.
2. Командний – чимось дуже схожий на приватний, але головною відмінністю є те, що доступ є тільки у певної групи людей, які були об'єднані в команду.
3. Дошка організації. Якщо говорити за цей рівень доступу, то він доступний в платній версії, все аналогічно до командного, але функціонал є більш розширеним для реалізації спецефічних питань.
4. Публічний – це дошки, які можуть використовувати будь-які користувачі, у яких є посилання, але змінювати можуть тільки люди які завтверджені за даною дошкою.

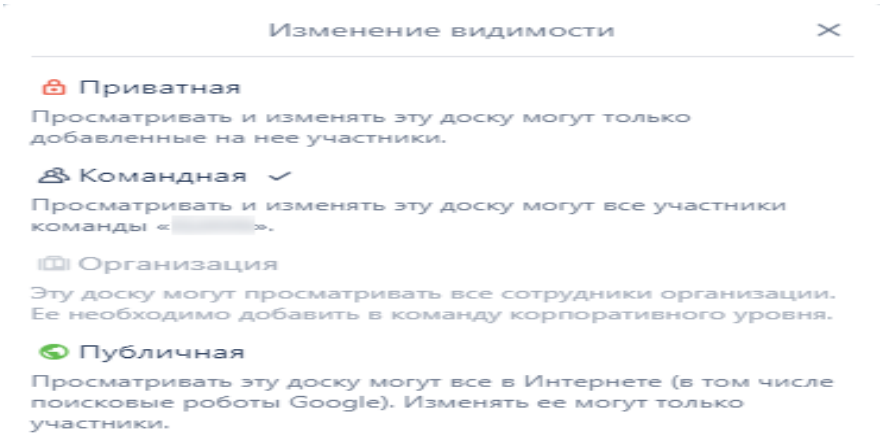


Рис 1.7. Рівні доступу до дошок сервісу Trello

Якщо говорити про мінуси, то:

- функціоналу Trello в повній мірі не буде вистачати для дійсно великих компаній, які маневрують від одної методики до іншої в пошуках оптимального та швидкого росту бізнесу
- на маленьких екранах, наприклад екранах телефону, Trello не є таким зручним
- складно працювати з автоматизацією та швидким додаванням завдань
- неможливо працювати, якщо немає доступу до інтернету
- відсутність української локалізації

Для ведення управління великих або як їх ще називають промислових проектів Trello не підійде. Адже не вийде повноцінно організувати з ним роботу цілої компанії, але коли мова йде про малі проекти, які не потребують специфічних інструментів, то дана система майже немає рівних в своїй сфері. Особливо він є гнучким з такими методологіями управління проектами такими як: Scrum, Kanban, які є популярними в сфері IT [6].

Максимальна ефективність даної системи розкривається при індивідуальній активності, відразу можна відмітити, що продуктивність помітно зростає. Ідеї та завдання швидко упорядковуються, все стає зрозумілим і наглядним, набагато легше розглядати проблему з розставленням пріоритетів, коли всі проблеми на одній дошці [5].

Картки мають безліч можливостей, їх можна примінити так як бачить керівник. За допомогою них можна вести безліч обговорень, голосувань, проводити завантаження файлів. Є можливість створювати дедлайни, виділяти інформацію за допомогою колірних міток. Також для будь-якого завдання можна затвердити виконавця. Для цього необхідно в картці вибрати його зі списку вашої команди. Приклад такого функціоналу зображений на рис. 1.8.

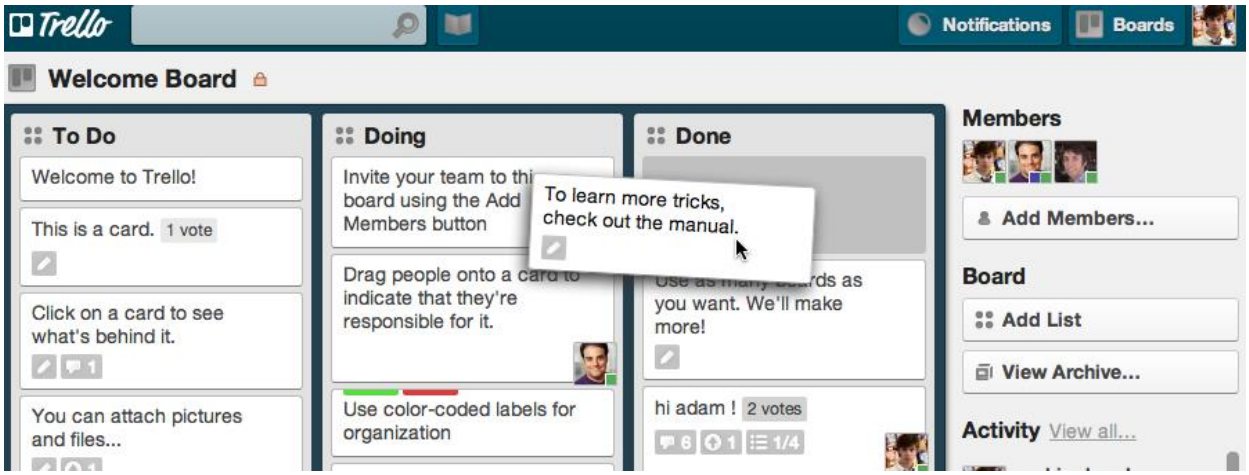


Рис 1.8. Головний екран Trello

Так як даний сервіс можуть використовувати багато людей одночасно, тут реалізовано ефективний механізм правки задач, а саме: зміни статусу завдань, присвоєння міток відповідального персоналу, за певну задачу. Одна з дуже важливих сучасних функцій це можливість переглянути історію. Це дозволяє відслідковувати хто вносив зміни і коли. Також присутня можливість проведення голосування для будь-якої задачі. Такий функціонал надає можливість дізнатись думку кожної людини з проекту і в кінці прийняти на основі цих голосів об'єктивне рішення [4].

Одним з найважливіших факторів, які необхідні на сьогоднішній день в подібних сервісах - це можливість виставляти дедлайни, адже у проектному управлінні важливі терміни виконання завдань. Для кожної картки можна налаштувати свій дедлайн. Він буде відображатись в ній під час перегляду списку. При бажанні можна встановити нагадування і тому з наближенням дедлайну на картці автоматично буде підсвічуватись дата.

Потрібно відмітити основні можливості Trello [6]:

- Інтерактивний інтерфейс
- Зручно робити оцінку всіх завдань відразу
- Можливість інтегрувати сторонні сервіси
- Завантаження та робота з файлами відразу на дошці
- Доступність на різних пристроях, так звана кросплатформенність

1.2.3. Any.Do

Any.Do - це такий сервіс, що дозволяє легко координувати справи по бізнесу, а також особисте життя відразу з одного інтерфейсу. Серед головних особливостей даної системи можна виділити хмарну синхронізацію та розпізнавання мови. Також присутня можливість налаштовувати час на виконання завдань і встановлення сповіщень. Головний екран даного ПЗ зображений на рис. 1.9.

Цей продукт має дуже цікаві і незвичні можливості в управлінні списками: для того щоб відмітити виконані задачі, достатньо буде просто стряхнути смартфон для їх видалення, така незвична функція актуальна лише для мобільних додатків. Можливість додавання різноманітних віджетів та розширень на головний екран надає цьому сервісу ще більшої гнучкості та ефективності взаємодії користувача та інтерфейсу.

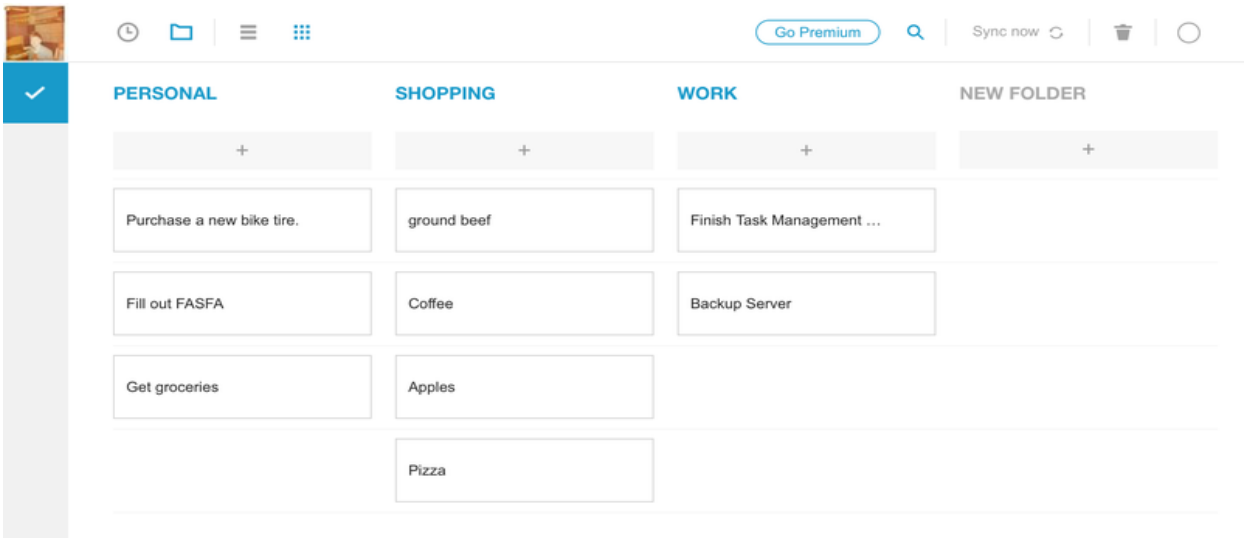


Рис. 1.9. Головний екран Any.Do

Дану систему можна використовувати як для індивідуального, так і для командного використання. Для цього наявний безкоштовний тариф. За допомогою якого можна ознайомитися з усіма функціями сервісу, в безкоштовній версії є обмеження на кількість пристроїв. В свою чергу преміум-версія передбачає установку на необмеженій кількості пристроїв, що розширить життя компанії.

Якщо говорити за підтримки додатку, то ним можна користуватись як і на комп’ютері, так і за допомогою мобільних пристроїв, новинкою даного продукту було введення розширення для Google Chrome. Це надало можливість використання системи навіть на тих пристроях, які не мають підтримки для десктопної версії продукту [7].

Особливу увагу потрібно приділити такому функціоналу, як голосовий помічник, даний функціонал зображений на рис. 1.10. Він буде цікавим тим людям, які живуть такому ритмі, що навіть не встигають виділяти час на друкування задач. Голосовий помічник надає можливість записувати ваш голос з подальшим його інтерпретуванням в текст задачі, все це буде оброблено самою програмою.

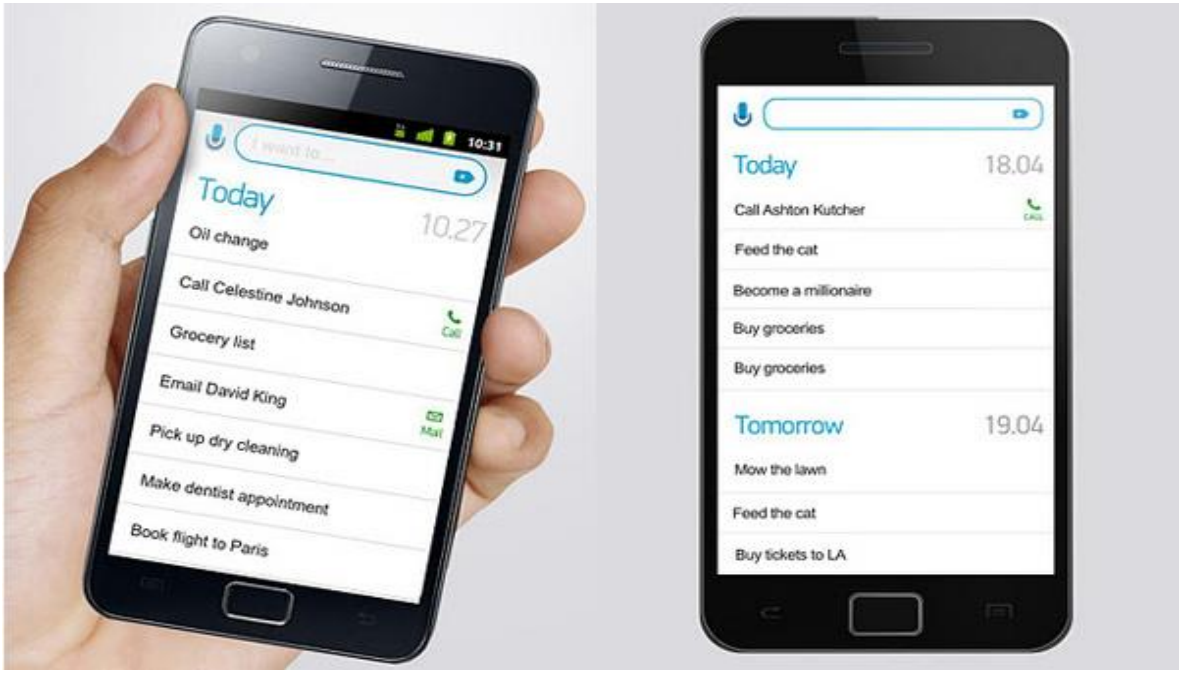


Рис. 1.10. Голосовий помічник в сервісі Any.Do

Any.Do надає можливість поділитись з іншими користувачами списком ваших справ, тобто це можливість налаштування прав доступу до списку завдань. Ви з легкістю можете відкрити його тільки для перегляду, чи не тільки для перегляду, а й для редагування усім хто має доступ. Це надає чудову нагоду та підвищує ефективність в повсякденних справах сім’ї чи в підтримці управління проекту в компанії, приклад цього зображений на рис. 1.11.

Any.Do - це зручний інструмент для формату тайм-менеджменту, адже при управлінні завданнями важлива скорість створення завдань і систематизація. Підведемо висновки якщо ви активно записуєте свої справи протягом дня і у вас не вистачає бажань набирати текст, а вам простіше записати голосове повідомлення, то дана система буде ідеальним рішенням для вас, без зайвих специфічних функцій, але з усім необхідним функціоналом під рукою [8].

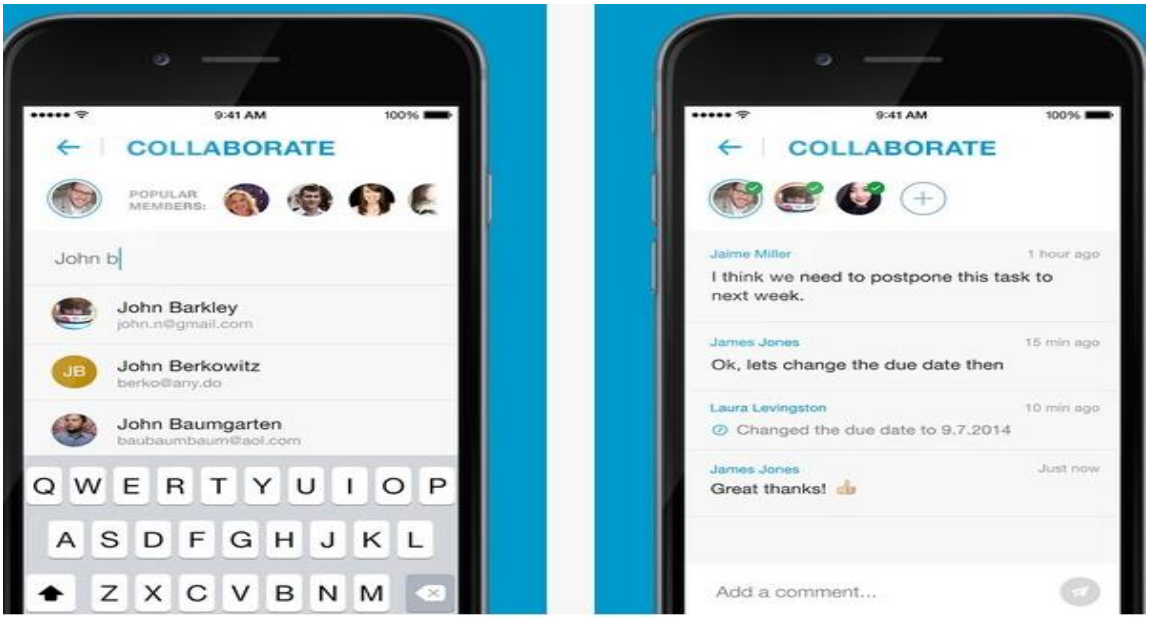


Рис. 1.11. Демонстрація налаштування колабораторів в Any.Do

Розглянемо основні можливості даного продукту [7]:

- ✓ Можливість використовувати дане ПЗ на різноманітних пристроях, тобто так звана кросплатформенність
- ✓ Створення задач та списків в своїх цілях
- ✓ Гнучкий та інтерактивний інтерфейс
- ✓ Можливість налаштування доступу для перегляду списків задач
- ✓ Інтерпретації голосових повідомлень в текстову задачу
- ✓ Налаштування дедлайнів та сповіщень про них

1.2.4. TickTick

TickTick – це зручний кросплатформенний тайм-менеджер з широкою функціональністю, головний екран TickTick зображений на рис. 1.12. Даний сервіс може синхронізувати задачі як з веб-версії та і з мобільних додатків та розширень для браузерів. Так як вже було сказано про кросплатформеність звернем увагу, що даний продукт може синхронізувати задачі як мобільного додатку, так і з веб-версії для різних браузерів.

Якщо говорити за мобільний інтерфейс, то тут, як і в більшості аналогічних сервісах він є досить зручним, більшість функцій запрограмовані під використання спеціальних жестів.

Так як TickTick дозволяє створювати велику кількість списків завдань, то саму систему використовують таким чином: в базовому вигляді існує можливість, створення трьох списків: робочі завдання, життєві завдання і просто замітки - це найпростіший сценарій використання даного програмного забезпечення [11].

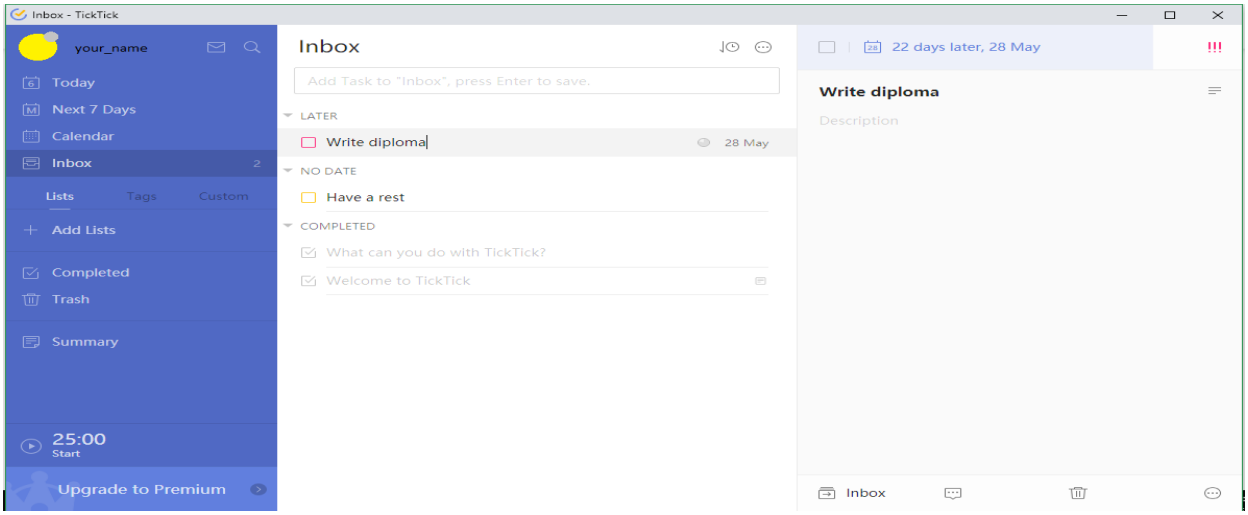


Рис. 1.12. Головний екран системи TickTick

Важливою перевагою даного сервісу – це можливість розділення списків по тематиці. Тобто в подальшому створення завдань в необхідній темі. Це допомагає з сортуванням питань між собою для швидкого реагування в робочих питаннях, домашніх справах та в плануванні довгострокових завдань.

Другий підхід, який розглядають в більшості випадків - це трирівнева система:

1. Першим рівенем будуть списки, які виступають набором проектів. Їх може бути велика кількість, адже все це зберігається на серверах TickTick і при бажанні ми можемо з легкістю убрати проект в архів і повернутись до нього коли в нас є можливість.
2. Другий рівень виступають завдання в будь-якому списку, а в нашому випадку проекту. Ці завдання ефективно та гнучко упорядковані відповідно до того які помітки ми встановим в фільтрах: по даті, по пріоритету, по назві тощо.
3. Третій рівень – це так звані чек-листи в рамках завдання, певні нотування до самого завдання.

Дана система надає можливість прив’язати задачу не тільки по даті, а й по геолокації, тобто ми налаштовуємо задачу на повторення в різний час та в різних місцях. Наприклад тричі на тиждень потрібно проводити зустріч з командою, для цього достатньо тільки один раз створити задачу і задати її повторення, як це зображено на рис. 1.13. Це ефективно вплине на наш час і позбавить рутинної роботи по створенню типових завдань[9].

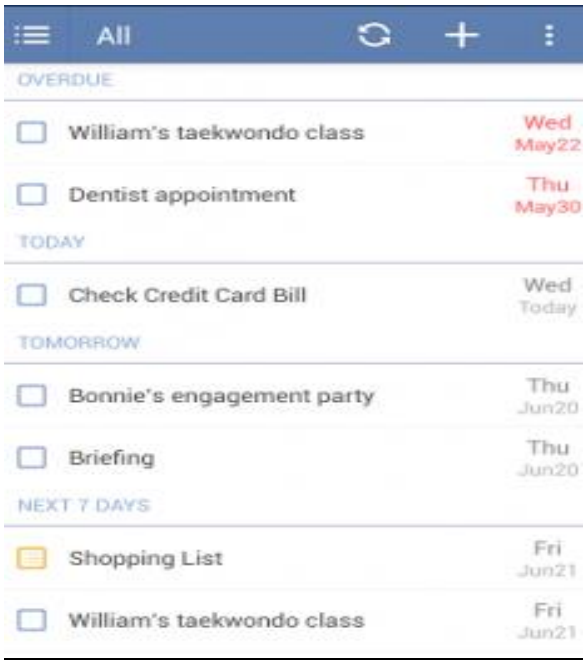


Рис. 1.13. Функція повторного створення завдань в TickTick

Як і в більшості аналогів даного ПЗ, що були наведені вище і окрім них, що є на ринку, дана система надає можливість поділитись списком ваших справ з іншими користувачами даного продукту, так як ми розглядаємо безкоштовну версію, то тут можна поділитись лише з однією людиною, для розширення даного функціоналу необхідно використовувати платну версію.

Не потрібно забувати, що сьогодні більшість сервісів мають можливість інтеграції, так і TickTick надає цю можливість з сервісом Google Now. За допомогою цієї інтеграції дозволяється додавати нові задачі, які створюються на основі інтеграції голосового повідомлення в текст задачі. З подальшим використанням сервісу Google Calendar, відбувається синхронізація задач в TickTick і час від часу прилітає нагадування в календарі.

Зрозумілий і легкий в використанні механізм створення міток задачам, забезпечує їх фільтрацію по міткам, дедлайнам тощо. Можна створювати коментарі до завдань та редагувати їх. Існує можливість додати для кожної задачі свій особливий пріорітет. Пріорітетів всього чотири: високий, середній, низький та відсутній, як зображено на рис. 1.14.

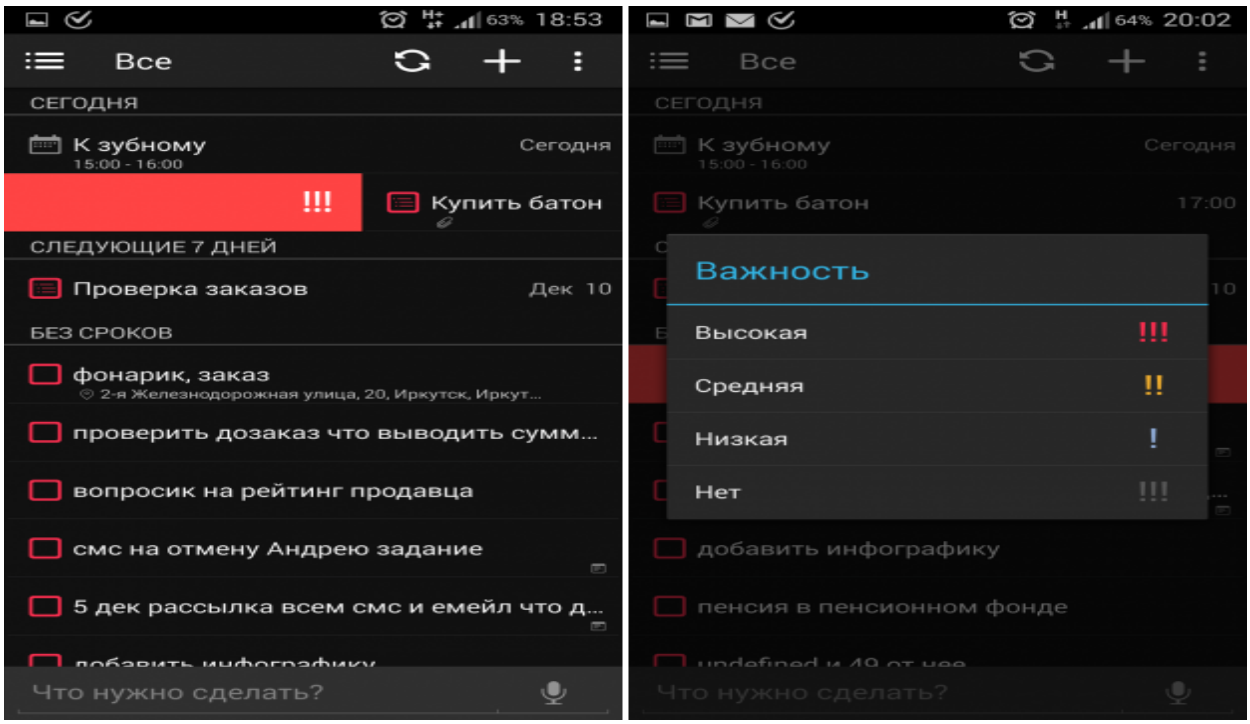


Рис. 1.14. Налаштування пріорітетів в сервісі TickTick

TickTick також дозволяє створювати нагадування про завдання з закріпленою геолокацією. Розглянемо випадок і повернемось до завдання купити хліб з прив'язаною геолокацією магазину. Мінімальний доступний радіус нагадування радіус 100 метрів є можливість його збільшення, приклад цього зображений на рис. 1.15. Особисто мені це дуже корисна функція, особливо для побутових дрібниць про які людина зазвичай забуває.

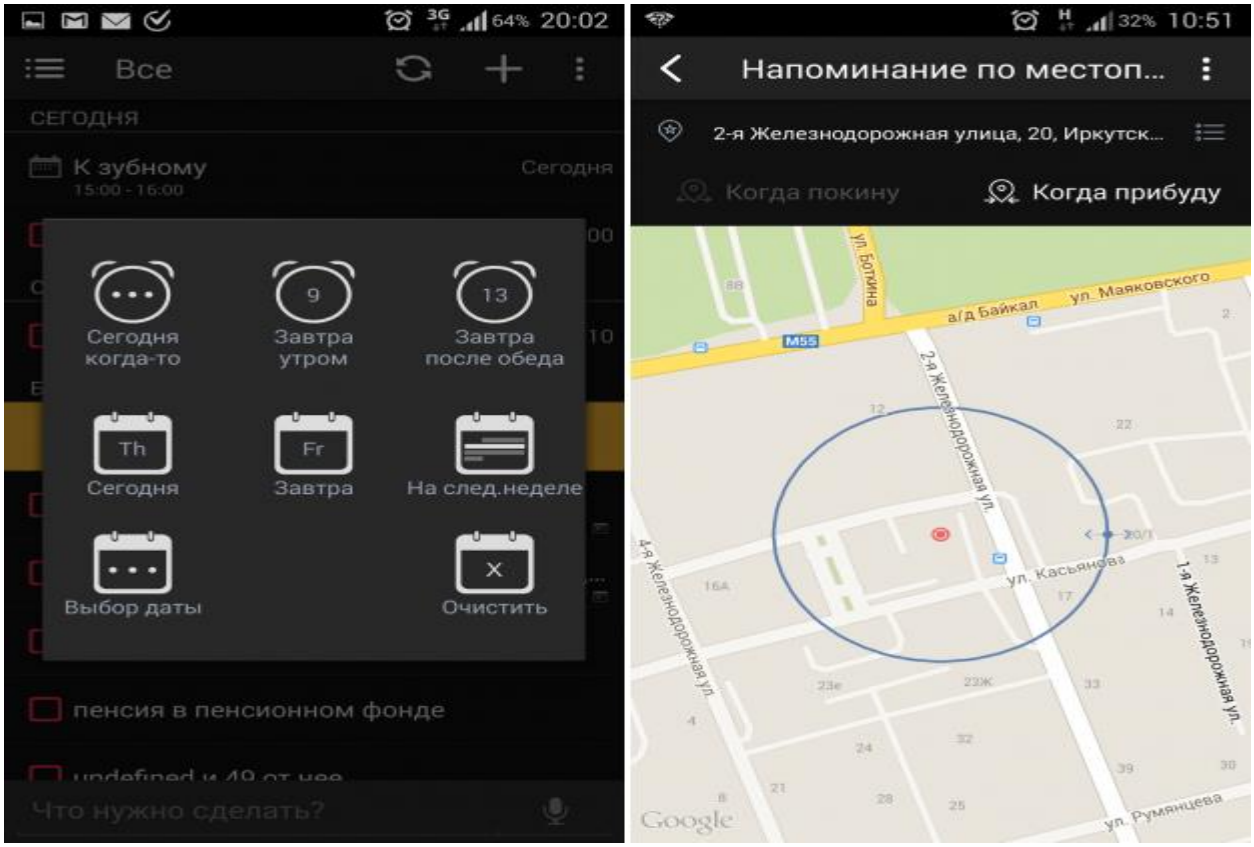


Рис. 1.15. Нагадування з використанням геолокації в TickTick

Перейдемо до основних можливостей системи TickTick [10] :

- Присутня можливість прив’язки задачі до геолокації
- Створення задач та списків
- Простий та інтуєтивно зрозумілий інтерфейс для взяємодії
- Присутня і мобільна версія даного продукту
- Можливість поділитись списком задач
- Інтерпретація голосового завдання в текстовий формат
- Повторне перевикористання задач
- Дедлайни та сповіщення

ВИСНОВКИ ДО РОЗДІЛУ 1

У даному розділі детально розглянуті та проаналізовані актуальні та новітні аналоги тайм трекінгу, які є сьогодні на ринку. Так як IT сфера не стоїть на місці, а розвивається з року в рік так і потреба в системах, які будуть допомагати в ефективному управлінні проектами зростає. Розглянуте програмне забезпечення в більшості випадків схоже одне на одного, але в певних моментах, воно може надати свій арсенал інструментів, для вирішення специфічних питань. Хотілось би відзначити, що основні тенденції сьогоднішнього прогресу в сфері тайм-трекінг або як його ще називають тайм-менеджменту спрямовані на прискорення швидкості обміну та взаємодії між даними та користувачами для отримання позитивних результатів. Проаналізовані дані наведені в таблиці 1.1.

Таблиця 1.1 Результати аналізу сучасних аналогів

Основна функціональність	Аналоги, які є на ринку			
	Jira	Trello	Any.Do	TickTick
Створення окремо взятих проектів	+	+	+	+
Наявність push сповіщень	+	+	+	+
Легке освоєння системи	–	+	+	–
Гнучкий та інтерактивний дизайн	+	+	+/–	+
Можливість прив’язки задач до календаря	+	+	–	+
Відстеження статусів проекту	+	+	–	+
Наявність пробної версії	+	+	+	+
Наявність мобільної версії	+	+	+	+
Ітеграція з іншими системами	+	+	+	+

Проаналізувавши таблицю вище, можем зробити висновки, що більшість додатків, якими наповнений ринок мають схожий функціонал, але виникають певні питання в зручності використання та спецефічному дизайні, адже це питання особисного характеру та ставлення. З розглянених сервісів найкращими виявились такі сервіси як: Jira, Trello, TickTick, кожен з цих сервісів відрізняється тим, що він заглиблений в свою проблематику і є доступність певних спецефічних та цікавих інструментів. У випадках коли розробники хотіли вмістити всі можливі інструменти в одній системі, як наприклад це зробити в Jira, даний продукт став більш складним в освоєнні.

Отже, при проектуванні та розробці власного програмного продукту в рамках дипломного проекту, слід орієнтуватись на сервіси Jira та Trello, які несуть стандартний пакет інструментів для вирішення питань з тайм-трекінгом. Звісно ж такий підхід є дуже суб'єктивним, адже він обґрунтований тільки власними враженнями від користання даних продуктів за короткий час.

На мою думку, при розробці програмного забезпечення, слід особливу увагу приділити дизайну та інтуєтивному освоєнню системи з можливістю зміни локалізації. Звичайно в мінорних версіях продукту, який проектується, функціонал може бути обмеженим, але так як тема з інструментами тайм-трекінгу є перспективна тому є сенс в подальшому розвитку .

РОЗДІЛ 2
ОПИС ТЕХНОЛОГІЙ ТА МОВ ПРОГРАМУВАННЯ ДЛЯ РІШЕННЯ ЗАДАЧІ

2.1. Опис завдання

Ринок та в цілому сфера IT на сьогоднішній день має дуже багато напрямків для розвитку і з кожним напрямком пов’язані свої мови програмування та технології з своїми принципами розробки для вирішення проблем в певній сфері. Повернемось до веб-додатків, одним з головних ідей побудови подібний додатків є проектування клієнт серверної архітектури, приклад такої архітектури зображений на рис. 2.1.

Дана архітектура містить в собі три основні компонента, такі як:

- Клієнт - до цього компоненту відносять те, що може сформувати і відправити HTTP запит на сервер
- Сервер – це такий компонент, який виступає як абстрактна ЕОМ в мережі, яка спроможна отримати HTTP запит від клієнта, обробити його і повернути йому коректну відповідь.
- База даних – цей компонент буде виступати сховищем для даних, ці данні зберігаються за певною схемою, яка лежала в основі проектування та всі дії та маніпуляції відбуваються за певними правилами [12].

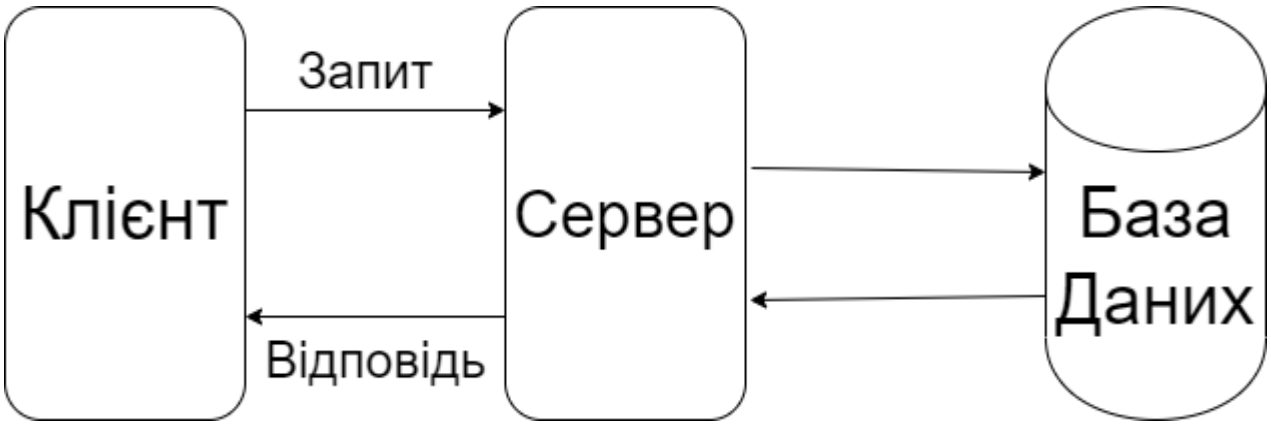


Рис. 2.1. Структура клієнт-серверної архітектури

В даному розділі будуть розглянуті та проаналізовані технології для розробки веб-додатків. Будуть описані всі необхідні технології для коректної

роботи кожного рівня клієнт серверної архітектури, а саме мова піде про такі технології, як:

- VueJS, AngularJS, ReactJS – найбільше використовуються сьогодні на ринку для створення програмного забезпечення, вони необхідні для створення клієнтської частини додатку.
- NodeJS, Java, Python – найбільш вживані для реалізації серверної частини продукту.
- PostgreSQL, HBase – дані бази даних використовують, як сховище даних в клієнт серверній архітектурі.

2.2. Node.js

Node.js – це така середа для розробки серверної частини на мові JavaScript коду, ця середа побудована на базі JS двигуна V8, який був розроблений компанією Google, на рис. 2.2 зображено схему роботи даної платформи. В 2009 році з'явився JavaScript і сьогодні він вважається серйозною мовою. Node.js завоювала популярність і стала лідером в сфері веб-розробки і широко використовується сьогодні на ринку. Найбільше Node.js використовують для написання веб-серверів, але цим все не закінчується [13].

Розглянемо основні переваги Node.js вони такі :

- **JavaScript –** можливо писати серверну та клієнтську частину не змінюючи мови, адже середовище Node.js виконує код на JavaScript.
- **Швидкість –** це забезпечується використанням неблокуючої архітектурою платформи на відміну від , таких мов як C++ або Java.
- **Двигун V8** – Node.js використовує в свому фундаменту JavaScript двигун V8. В основі V8 компіляція коду відбувається минаючи стадію байт-коду, також за допомогою нього можна визначати, де знаходяться в пам'яті об'єкти, все це дозволяє запобігти

витоку пам'яті при помилках ідентифікації об'єктів в якості роботи покажчиків.

- **Асинхронність –** як ми звикли, що в сучасних реаліях і розвитку IT мовам програмування є звичним факт про процеси, що виконуються є самі блокуючими. Щоб написати асинхронне виконання, нам потрібно реалізовувати це самостійно. В JavaScript присутня можливість написання асинхронного коду для цього можна використати поток функцій зворотнього виклику. Його ще називають callback function, ця реалізація заснована на взаємодією з подіями в JavaScript. Цей принцип працює так : відбувається передача функції зворотнього зв'язку у спеціалізований механізм, він буде примінений відразу коли операція буде завершена. Дана функціональність надає величезні можливості серверу обробити безліч запитів. І головним плюсом являється те, що розробник не хвилюється про проблеми створення та взаємодії потоків [14].

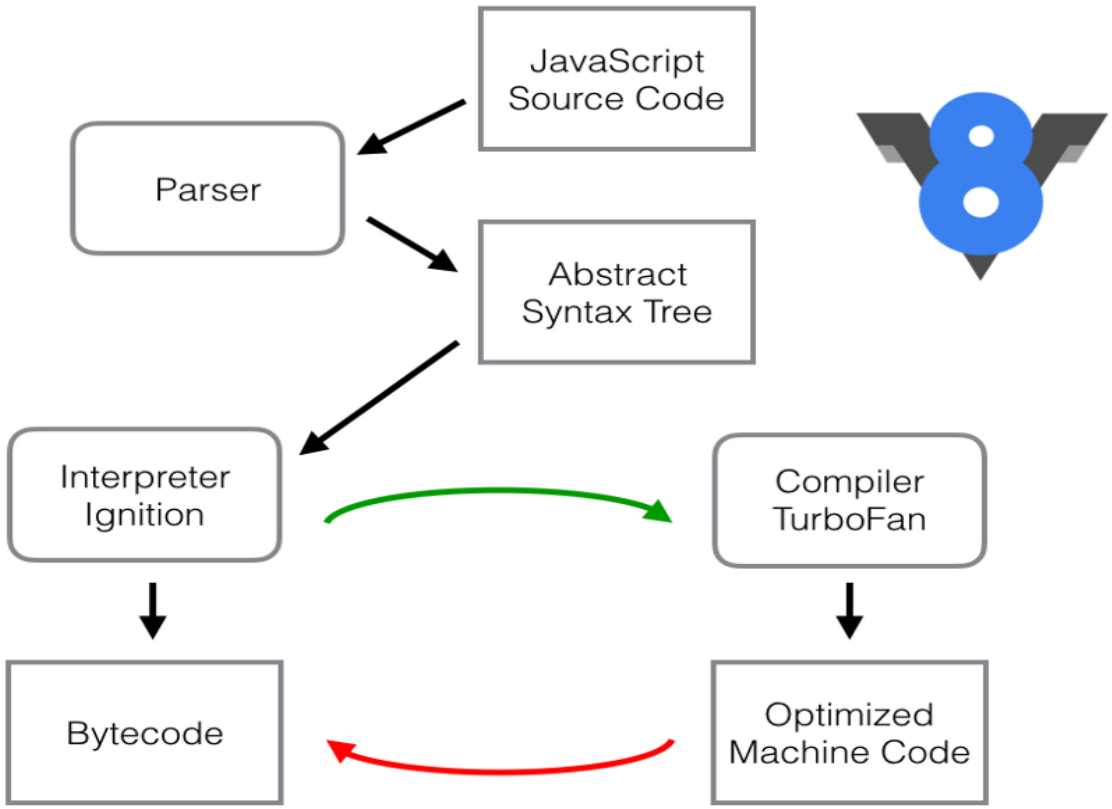


Рис. 2.2. Виконання JS коду на платформі Node.js

Також не варто забувати, що для зручної роботи є багато бібліотек, таких як:

1. NextJs – дана бібліотека використовується для серверного рендеринга коду на React.
2. Express - вважається найпотужнішою з універсальних фреймворків для серверів.
3. Micro – вважається легкою і компактною, бібліотекою для HTTP-мікросерверів.
4. Koa – нащадок бібліотеки Express. Даний Фреймворк є потужним, але при цьому більш компактним в розмірі.
5. Socket.io – бібліотека яку використовують для реалізації питань з системами реального часу [13].

Для ефективної розробки, було запропоновано механізм node version manager, він є досить зручним і допомагає встановити декілька версій відразу. Тобто для кожного проекту ми використовуємо різні версії NodeJs , без синхронізації, а просто зберігаєм середовище в якому було створено проект. Дана платформа є дуже потужною та гнучкою. Якщо є бажання створити власний продукт знаючи тільки JavaScript, то вибір очевидний.

2.3. Java. Spring framework

Мова програмування Java на сьогоднішній день є однією з популярних мов, які використовують для написання програмного забезпечення, взагалі ця мова є об'єктно-орієнтована і була розробла компанією Sun Microsystems. В подальшому її було придбано компанією Oracle, яка володіє її правами по сьогоднішній день. Перший реліз даної мови світ побачив 23 травня 1995 року, логотип Java зображений на рис. 2.3.

На початку зіркової кар’єри, цю мову програмування називали як Oak і розроблялась вона для побутової електроніки. Згодом Oak був перейменований в мову Java, якою вона і є на сьогоднішній день. Спочатку, коли Java починала свої перші кроки її використовували для написання аплетів. Основною ідеєю даної мови було створення програм на Java, які

можуть трансльовані, тобто переведені в байт-код. В свою чергу байт-код виконується віртуальною машиною, або як її назвають JVM. Віртуальна машина, яка опрацьовує байт-код і передає інструкції обладнанню. Тобто вона виступає як інтерпретатор, але головною відмінністю та ідеєю є те , що байт-код, на відміну від звичайного тексту, обробляється значно швидше.

Рис. 2.3. Логотип Java

В часи, коли Java ще не була створена, більшість проектів писалось на мові C чи C++. Актуальною була C++, адже вона також є об'єктно-орієнтована і могла вирішити необхідну проблематику. З часом росту технологічного процесу, виникала й проблема кросплатформенності, тоді для вирішення даної проблеми було розроблено платформонезалежну мову. Все це означало, що така мова надає можливості створювати програмне забезпечення, яке не потрібно компілювати для кожної архітектури окремо. Просто можна використовувати таке програмне забезпечення навіть на різних процесорах з використанням різних операційних системам. Саме тому Java перейняла багато корисних функцій від C++ і створила нові можливості в розвитку технологічного процесу світу.

Java об’єднала три основні концепції того часу:

1. Широкий спектр використання аплетів - це невеликі, але надійні на той час, мережеві додатки, які вбудовувались в веб сторінки, головной їх перевагою було простота налаштування.

2. Підтримка об'єктно-орієнтованої моделі розробки додатків, яке надає можливість поєднати вже всім знайому логіку побудови додатків спираючись на реальне представлення речей.
3. Надає великий вибір інструментів, одиницею виступає звичайний клас за допомогою якого будується складна архітектура, яка несе якусь бізнес-логіку .

Наведемо основні переваги Java:

- Інтерпретованість – як згадувалось вище, Java опрацьовує байт-код й переводить його в машинні інструкції.
- Об’єктно-орієнтованість – надає більш реального уявлення сфері та сутностям з якими працюють.
- Кросплатформенність – працює не тільки з однією апаратною платформою, тобто компіляція не на платформі конкретної машини, а за допомогою використання платформи байт-кода.
- Багатопотоковість – можливість створювати багатопотокові додатки для оптимізації та прискорення роботи програми.
- Високопродуктивність – все це забезпечуться JIT компіляторами, вони побудовані на концепції збільшення продуктивності додатків, що використовують байт-код. Це все досягається за допомогою компіляції байт-коду в машинний код під час роботи програми.
- Архітектуро-нейтральність – цей принцип забезпечує можливість доступу до коду в Runtime, це досягається через те що компілятор створює архітектурно-нейтральні об’єкти у вигляді файлу [15].

Java на сьогоднішній день є дуже потужним інструментом для створення програмного забезпечення. Також розробникам надається багато різних фреймворків та бібліотек, які прискорюють розробку продукту та роблять її більш ефективною, надаючи вже готові рішення. В сфері веб розробки зв’язаний з Java являється фреймворк Spring.

Spring Framework – це фреймворк в Java, тобто це певний каркас, який надає можливість використовувати вже готові рішення для вирішення концептуальних проблем побудови веб-додатків. Іншими словами це виглядає так: це створення свого класу з певною логікою, а вже створення самого об'єкту вашого класу і подальші дії за вас це робить сам фреймворк. До створення даного фреймворку розробка велась так: створений клас імплементував якісь інтерфейси з фреймворка, ось так отримувалась вже певна готова логіка необхідна для роботи сервісу. Але в фреймворку Spring захотіли відійти від цієї концепції і для цього ввели анотації, які зменшили написання лишнього коду в рази, що прискорило швидкість написання веб-додатків в сфері IT.

Однією з ідей в даному фреймворку це принцип IoC. Він забезпечує надання залежності і управління зовнішнім принципом, це призводить до того, що код є менш зв’язним. Реалізацією даного принципа є Dependency Injection і Dependency Lookup. В свою чергу Dependency Injection можна забезпечити через використання таких можливостей мови як:

1. Використання конструктора
2. Через поле
3. За допомогою використання метода в ролі сетора.

В свою чергу Dependency Lookup працює таким чином:

1. Вивільнення залежності JNDI, тобто в нас є залежний об’єкт, відбувається пошук в реєстрі JNDI і з контейнера отримуємо потрібну залежність.
2. Контекстний пошук все аналогічно до попереднього принципу, але цей працює скоріше, адже залежність відразу шукається в контейнеру без реєстру JNDI.

Для розуміння картини, як все ж таки влаштований даний фреймворк його структура представлена на рис. 2.4. Як видно з даного рисунку Spring має модульну структуру, це надає можливість підключити і використовувати

тільки ті модулі, які нам потрібно. В свою чергу даний фреймворк можна поділити на три рівня:

- Data Access – надає функціонал роботи з даними в основному це робота з БД.
- Web – надає головний функціонал для створення веб-додатків, сама ідеологія
- Core – це ядро, тобто реалізація концепцій Spring.

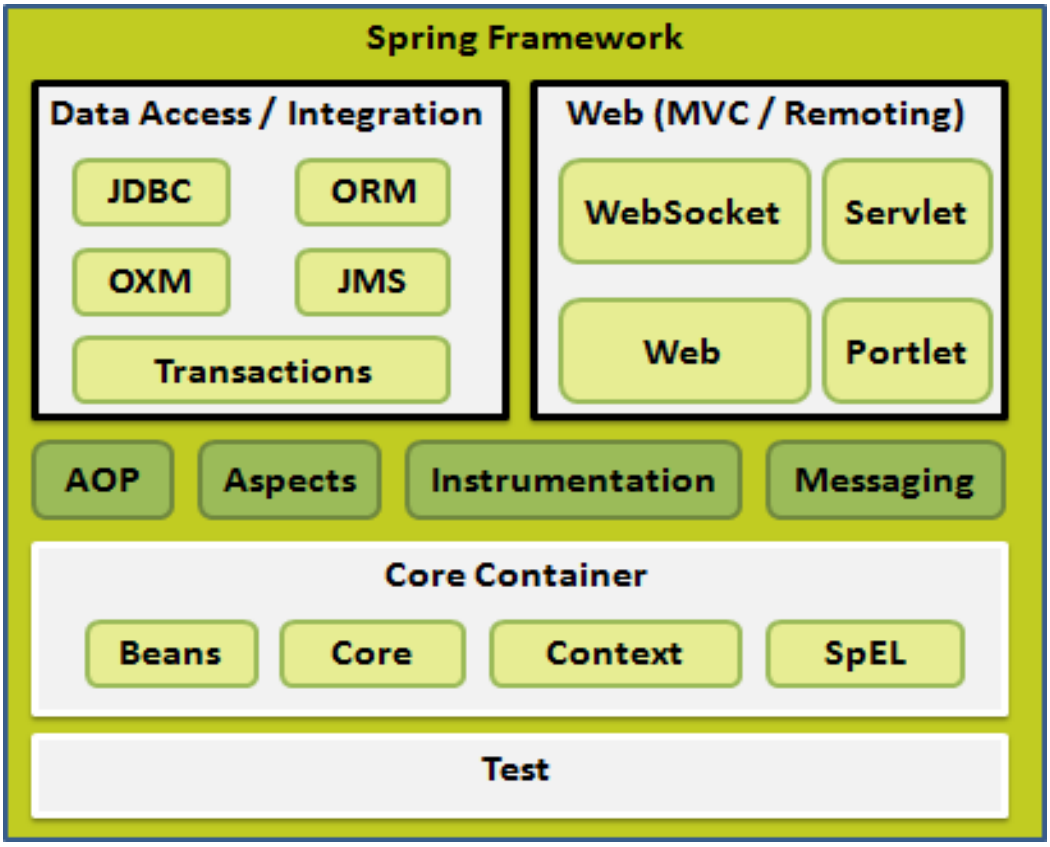


Рис 2.4. Структура фреймворку Spring [16]

Отже, це потужний фреймворк в який входить ще багато інших проектів, так як: Spring Boot, Spring Data, Spring Security, Spring MVC, які допоможуть в ефективній розробці продукту.

2.4. Python.Django

Фреймворки як завжди полегшують життя розробника, пропонуючи їм вже готовий каркас для розробки додатків. І зараз мова піде про такий фреймворк як Django – данний фреймворк є безкоштовним з відкритим кодом. Ось певні особливості, які притаманні для даного фреймворку це

вбудована автентифікація, маршрутизація URL-адрес, механізм шаблонів , об'єктно-реляційний маппер та міграція схеми бази даних.

Дані функції надають можливість створювати додатки маштабованими, швидким та надзвичайно універсальним. Django має можливість використовувати ORM для відображення об'єктів у таблиці баз даних. Код буде працювати з різними базами даних, розробник не матиме трудностей з перенесенням з однієї бази даних в іншу. Django працює з такими базами даних як: PostgreSQL, MySQL та Oracle, проте сторонні драйвери дозволяють співпрацювати з різнотипними базами чи то колонкові, графові, ключ-значення, документоорієнтовані.

За опитуванням 2019 року Python використовують 31.2 відсотки розробників, це зображено на рис. 2.5.

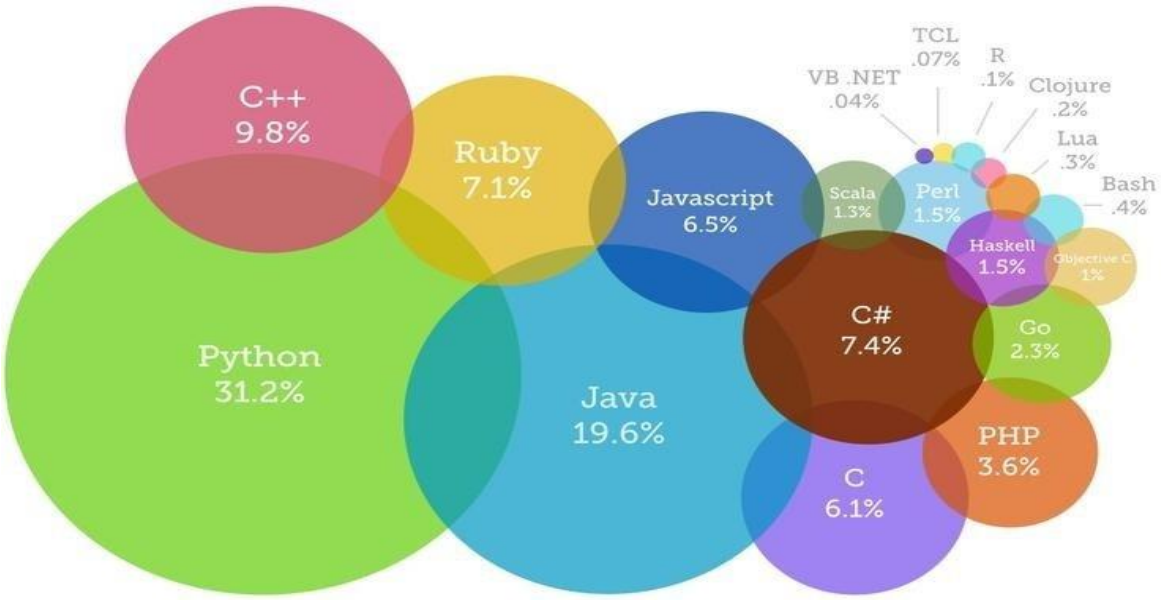


Рис 2.5. Статистика використання мов за 2019 рік

Розглянемо плюси Django:

- Повна комплектація – це означає, що є можливість працювати з десятками додаткових функцій. До цього можна віднести такі речі: аутентифікацією користувача, карти сайту, адмініструванням вмісту, RSS тощо.
- Швидкість – даний фреймворк був розроблений для того, щоб створювати додатоки настільки швидко, на скільки це можливо. Ідеєю

розвитку даного формату було подолання проблем з дедлайнами це надихнуло, створити такий продукт, який максимально прискорив би розробку.

- Безпека. Дуже цікавий пункт, адже при розробці, є допомога для захисту від помилок, пов'язаних з безпекою, які можуть ставити під загрозу цілісінький проект. До цього можна віднести такі поширені помилки, як: SQL ін’єкції, крос-сайт підробки і крос-сайтовий скриптинг, тобто різноманітні взломи, які широко поширені на ринку. Для того, щоб вести ефективне використання логінів і паролів, система надає різноманітні ключі для аунтентифікації.
- Масштабованість – це означає, що даний фреймворк найкращим чином підходить для роботи з великою завантаженістю. Багато сайтів з високим трафіком використовують таку можливість оптимізації.
- Різнобічність – всі питання в сфері взаємодії з контентом, чи якісь обчислювальні платформи все це можливо створювати за допомогою даного фреймворку [17].

Існує дуже багато плюсів і мінусів стосовно використання Django. Але відверто кажучи, коли розглядається веб-проект з поспішними дедлайнами, де результат грає велику роль, то використання Django – це дуже правильне рішення. Також для прискорення робочого процесу, можна просто встановити кастомну конфігурацію. З використанням даного фреймворку сама вартість розробки почала знижуватись.

2.5. ReactJs

React або ReactJS даний фреймворк призначений для веб-розробки клієнтскої частини. Розглянемо основні концепції, на який побудований ReactJS:

- Компоненти – до цього можна віднести елементи, як користувацький інтерфейс, адже він має свою власну функціональність й запрограмовану структуру.

- Елементи – це всім звичні об’єкти JavaScript. Дані об’єкти представляють HTML-елементи та описують DOM-елементи або так звані Document Object Model, приклад такого дерева наведений на рис. 2.6.
- JSX – це така технологія для того, щоб створювати різноманітні елементи та компоненти в ReactJs. Цей синтаксис, схожий на XML, який також використовується в React. Даний синтаксис використовує препроцесор, щоб переробити HTML-подібний формат в стандартні об'єкти JavaScript. Дані об’єкти використовуються для аналізу движком JavaScript.

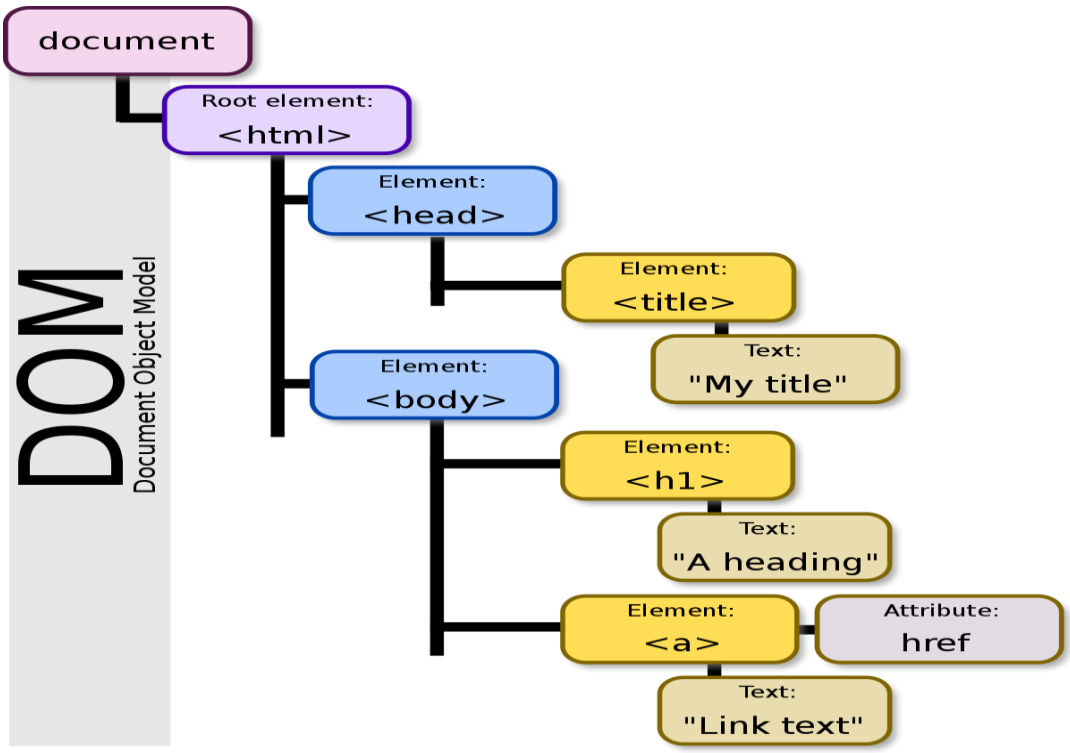


Рис. 2.6. Приклад DOM-дерева в ReactJs

- Virtual DOM – дане дерево React елементів використовується в браузері і це робить інтерфейс видимим. Під час роботи ReactJs детально аналізує та відслідковує всі зміни, які відбуваються в Virtual DOM. Ну й автоматична зміна DOM в браузері відбувається так, що той відразу відповідає віртуальному, демонстрація даної взаємодії зображена на рис. 2.7.

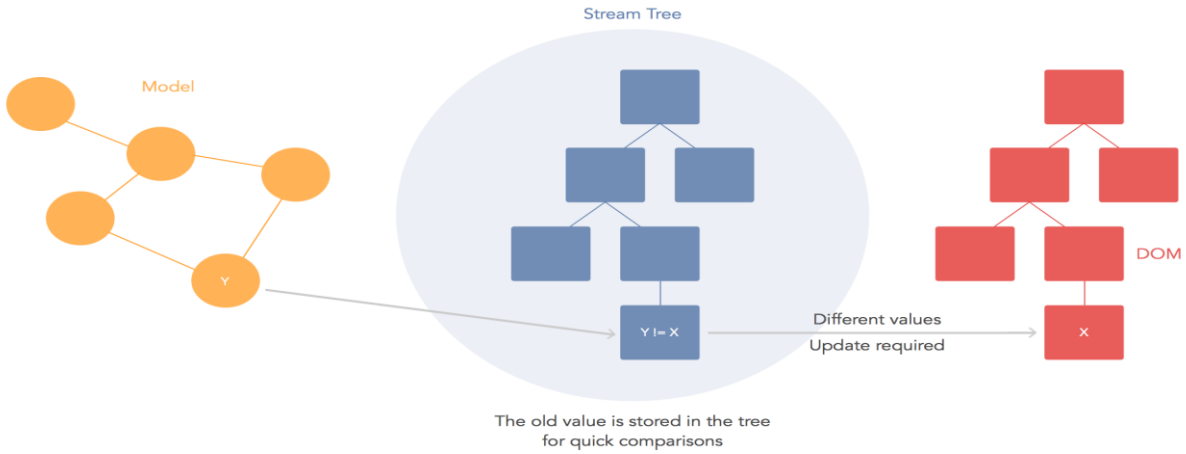


Рис. 2.7. Схема відображення DOM-дерева в ReactJs

Найхарактерніша і продуктивна річ в ReactJs це компонентний підхід створення. Розберемо більш детально, кожен з спроектованих компонентів має такі характерні налаштування, як стан та властивості. Відразу, коли відбувається відображення компонентів, все будується так, як його описано за допомогою JSX з врахуванням налаштувань: стану та властивостей. Під час виконання є можливість звичайно змінити властивості. Для цього використовують спеціалізовану функцію і тоді Virtual DOM приймає всі зміни стану і відбувається перебудова компонентів з подальшими змінами в веб-інтерфейсу.

Всі розробники, які користуються даним компонентно спрямованим підходом дуже задоволені. В цілому він заключається в тому, що для початку просто достатньо створити один компонент. Даний компонент буде приймати властивості при створенні та перевикористовувати їх безліч раз. Декілька компонентів, які можна згрупувати, можна використовувати при створенні більшого та новітнього компоненту, ця вся ідея та даний підхід дістали назву - композиції компонентів. Тобто в нас виникатиме ієрархія компонентівь, вона схематично зображена на рис. 2.8. Розробники надали можливість передавати дані між батьківськими та дочірніми компонентами для цього просто потрібно задати властивості.

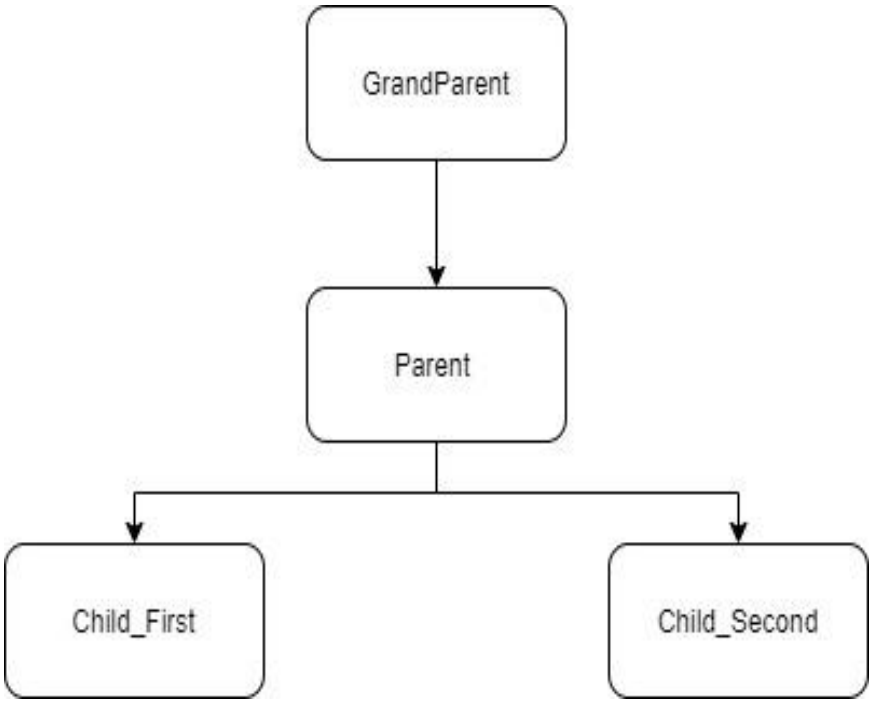


Рис. 2.8. Ієрархія компонентів React [18]

В цілому ReactJs все ж таки потужний і також в свою чергу легкий інструмент для веб-розробки, недаремно його плутають і називають бібліотекою. Так як в даному фреймворку використовують компонентний підхід, то все це робить даний інструмент дуже гнучким. Також велику увагу потрібно приділити розумінню використання віртуального DOM дерева, яке є досить швидкимі прискорює розробку програмного продукту. Також потрібно відмітити, що дана технологія в багато разів простіша у розумінні та вивченні.

2.6. VueJs

VueJs - це фреймворк з лінійки JS, призначений для розробки клієнстської частини програмного продукту. Перша версія даного фреймворку була випущена на ринок в лютому 2014 року. Даний фреймворк є досить популярним на рівні з ReactJs, вони вважаються найкращими в лінійці фреймворків Javascript. Але різниця звичайно присутня, адже вони надають нам різні особливості та функціональність. Основна відчутна відмінність VueJs від ReactJs полягає в тому, що VueJs використовує шаблони з декларативним візуалізацією. В свою чергу ReactJs використовує технологію JSX, про яку говорилось вище.

Перейдемо до головних функції VueJs:

- Прив’язка даних – прив’язка даних надає можливість керувати значеннями або просто просвоювати їм HTML-атрибути, а також редагувати стиль. За допомогою використання директиви v-bind можна присвоювати класи.
- Компоненти – даний підхід вже згадувався вище, коли мова йшла про ReactJs, в свою чергу VueJs все зав’язано на компонентному підході також. Даний підхід надає можливість створити елементи користувацького інтерфейсу з подальшою можливістю їх багаторазового використання. Компоненти проектуються за стратегією розбиття інтерфейсу, тобто відбувається розбиття на цілісні елементи, і далі відбувається композиція даних елементів, приклад даного принципа зображений на рис. 2.9

Взаємодія між цими компонентами відбувається за принципом «Props is, Events out». Цей принцип говорить про те, що існує можливість від батьківського елементу до дочірнього передати інформацію через задані властивості. І вже при передачі в зворотному напрямку буде викликатись сама подія, для більшого розуміння цей принцип схематично зображено на рис. 2.10.

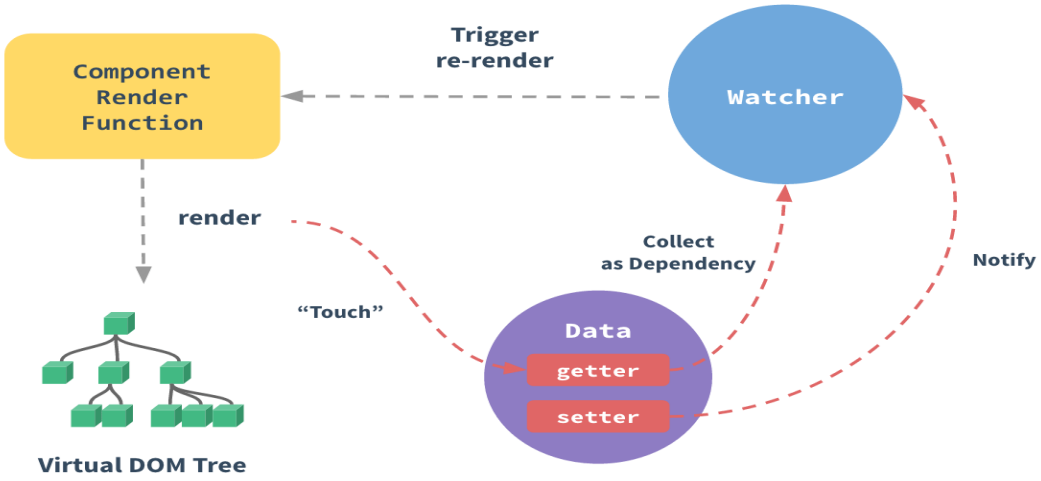


Рис. 2.9. Схематичне зображення розбиття інтерфейсу на компоненти

- Virtual DOM - VueJs використовує віртуальне DOM-дерево так само як і ReactJs. Тільки в даному випадку вся робота ведеться з копією основного дерева. Коли зміни були зроблені, віртуальне дерево починає перебудовуватись. Після закінчення перебудови всі його дані порівнюються з основним DOM деревом тільки тоді весь його інтерфейс перебудовується. Використання даного підхіду дуже корисно використовувати для оптимізації. Головною причиною цього є те, що він не вимагає великої кількості ресурсів і все це призводить до того, що зміни будуть відбуватись швидше.
- Переходи – це окрема тема, яка підкорила розробникі, що використовують VueJs. Дана функціональныість надає можливість добавити різні анімаційні ефекти до вашого програмного продукту. VueJs надає такі цікаві рішення як:
  - ✓ Створений автоматичний застосунок для класів, який виступає для створення і корегування CSS-переходів та анімацій.
  - ✓ Доступна можливість інтеграція різноманітніх сторонніх бібліотек для CSS-анімацій.
  - ✓ Можливість використання JavaScript для різноманітних маніпуляцій з DOM-деревом.

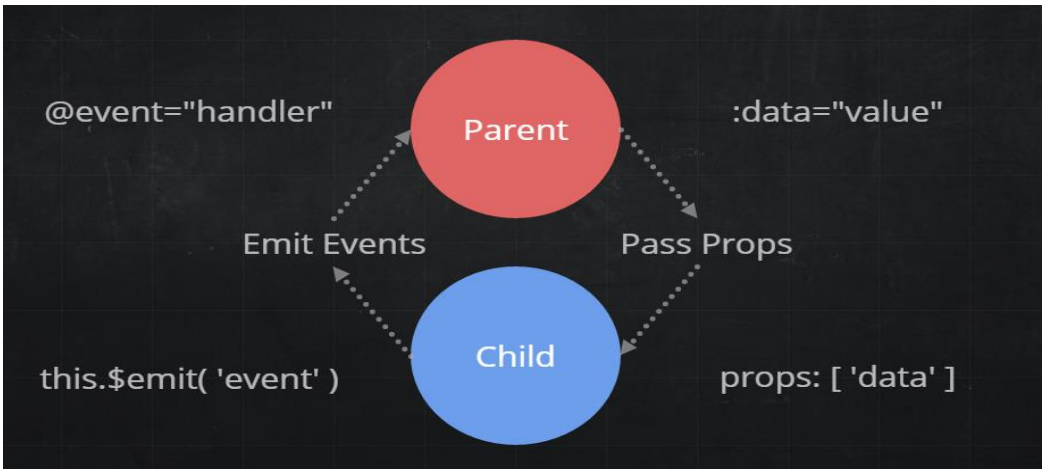


Рис. 2.10. Схематичне зображення принципа «Props is, Events out» у VueJS

- Управління станом компонента – кожен розробник, який використовував VueJs знає бібліотеку Vuex. Дана бібліотека надає централізований стан, тобто він є спільним для всіх компонентів. А також забезпечує всі правила, які генерують передбачувані зміни стану. Для наглядно розуміння на рис. 2.11 зображено приклад такого додатку з використанням Vuex. Розберемо детальніше, даний додаток містить такі наступні частини як:
    - ✓ State – це є головним і єдиним джерелом даних для самого компоненту.
    - ✓ Actions – дії, працюють з різноманітними подіями, які відбулись. Вони надходять і із різноманітних зовнішніх API і запускають потрібні процеси реагування.
    - ✓ Vue-components – звичайні компоненти про, які мова велась вище.
    - ✓ Mutations – використовується для змінювати станів, процес відбувається таким чином: коли ми отримали дані від частини Actions, ми просто змінюємо їх на стані.

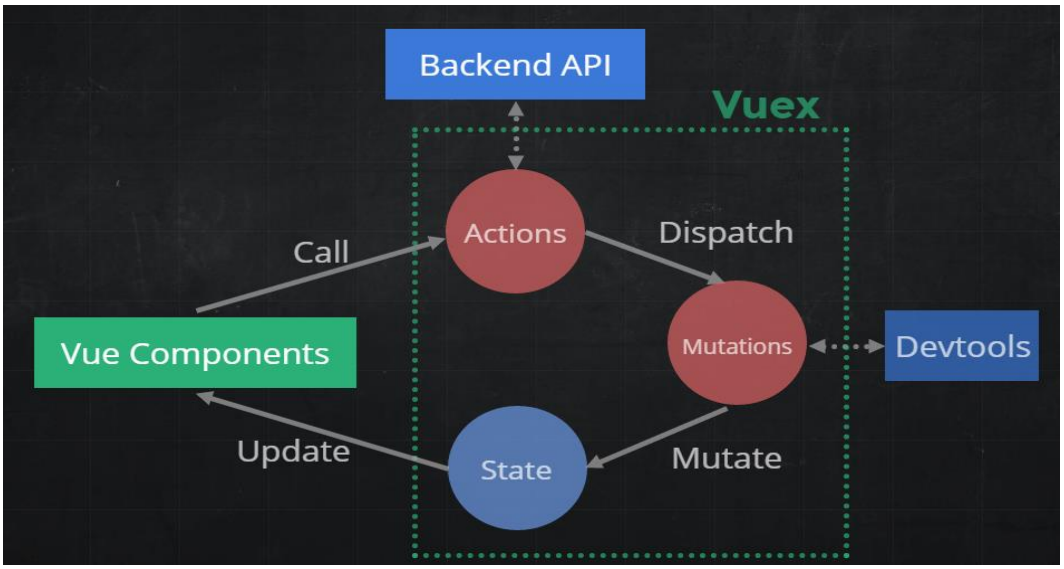


Рис. 2.11. Схематичний приклад управління станом у VueJs [19]

2.7. AngularJS

AngularJs – це JavaScript фреймворк з відкритим кодом для створення і підтримки різноманітний веб-додатків. Надає можливість використовувати HTML для шаблонів, також розширює синтаксис, все це робить код більш лаконічним і підвищує його ефективність та гнучкість.

AngularJs був спроектований для того, щоб створювати SPA сторінки. Дані сторінки надають таку можливість, як створення двостороннього зв'язування. Дане зв'язування надає можливість динамічно змінювати дані в одному місці інтерфейсу в той час як відбуваються маніпуляціях з даними моделі в другому. Цікавим рішення є те, що Angular використовує TypeScript для програмування.

Даний фреймворк має змогу використовувати такі механізми як: Dependency injection про який вже згадувалось, коли йшла мова про Java фреймворк Spring, а також Data-binding. Ці механізми спрощують розробку й зменшуються кількість коду в багато разів. Приклад роботи Dependency injection зображений на рис. 2.12.

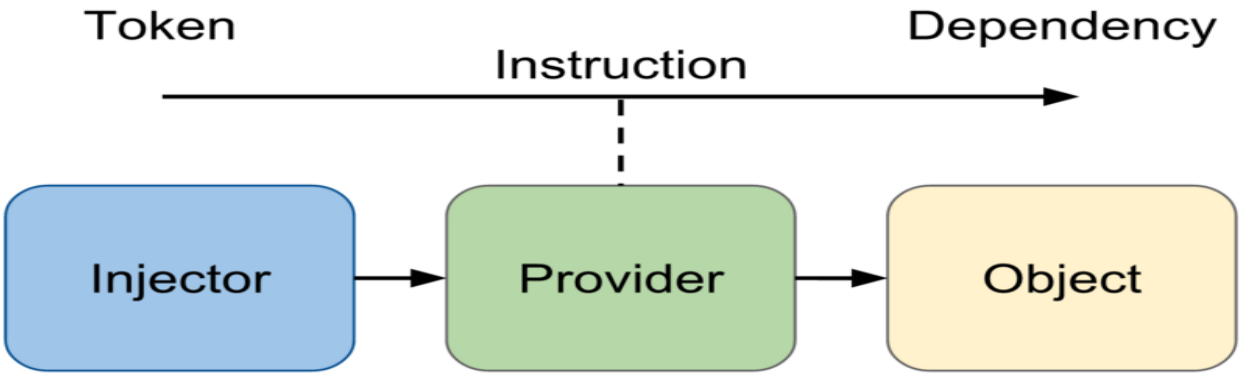


Рис. 2.12. Реалізації Dependency injection в AngularJs

Як згадувалось раніше даний механізм надає можливість впровадити залежності, тобто є можливість сворювати склідні об'єкти, які окрім полів містять силки на інші об'єкти. Все це створення відбувається автоматично, розробник повинен вказати те, що має впровадитись, прописавши це в конструкторі компоненту і на етапі компіляції фреймворк зробить всю роботу за нас. Розберемо більше детально, що таке Data-binding. Він допомагає з вирішенням проблеми прив'язки даних. Тобто він надає таку

можливість зробити прив'язку певних частини шаблону до конкретних визначених значень, які прописані в компоненті, даний принцип роботи зображений на рис. 2.13.

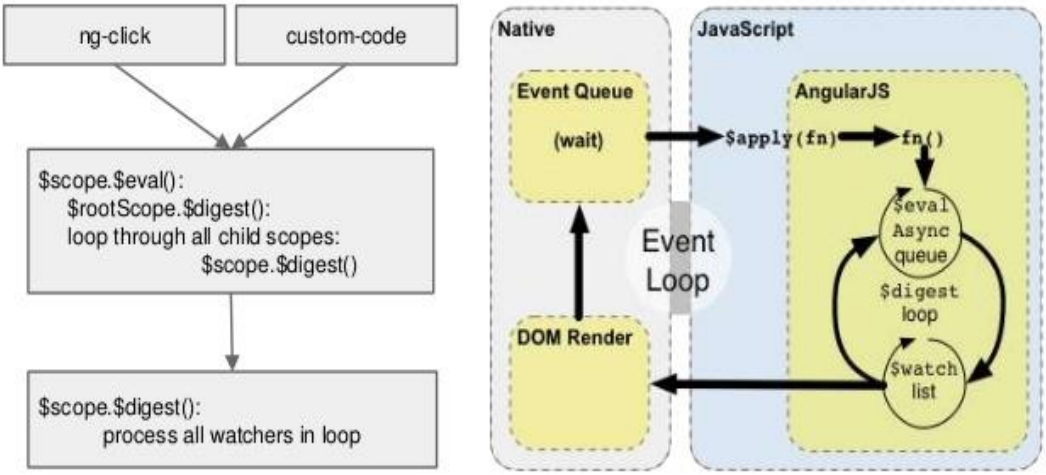


Рис. 2.13. Принцип роботи Data-binding в AngularJs [20]

AngularJs це дуже потужний та гнучкий фреймворк, який створений для написання SPA сторінок в сфері веб-розробки. Даний фреймворк використовується для написання клієнтської частини, потужні різноманітні механізми дозволять спростити розробку й зменшувати кількість написаного коду. Для сприйняття даний фреймворк важкий і тому не рекомендується його вчити якщо, немає досвіду в програмуванні і в створенні клієнтської частини в цілому.

2.8. PostgreSQL

PostgreSQL – це об'єктно-реляційна система, яка була створена для управління базами даних. Як аналогом може виступати MySQL. Дана база даних була розроблена для роботи на UNIX-подібних платформах, хоча сьогодні не виникає проблем при роботі і на таких платформах як: Mac OS X, Solaris та Windows. Дана БД вимагає дуже мінімальних зусиль, через свою стабільність, головний екран зображений на рис. 2.14. Це впливає на те, що при розробці програмного продукту з використанням даної БД, загальна вартість володіння даним продуктом є дуже низькою.

Розглянемо основні функції, які надає PostgreSQL:

1. Типи, які можуть задаватись розробником
2. Можна використовувати спадкування таблиць
3. Удосконалений замикаючий механізм
4. Зовнішня ключова референтна цілісність
5. Вкладені транзакції або як їх ще називають точки збереження
6. Багатоверсійний контроль сумісності
7. Асинхронна та синхронна реплікація

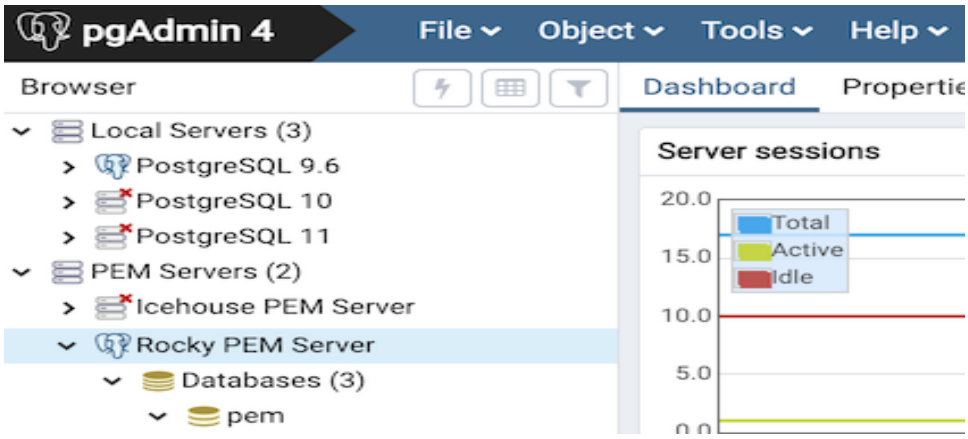


Рис. 2.14. Головний екран PostgreSQL

Головною особливістю даної системи – це управління базами даних, в якій реалізована функція контролю над одночасністю багатоверсійної версії. Дана функція носить назву MVCC і вона відома як ізоляція знімків у Oracle. Так дана БД надає можливість додавати спеціальні функції, розроблені за допомогою різних мов програмування, таких як C++, Java тощо. PostgreSQL надає можливість визначити власні типи даних, функціональні мови і подібні речі, все це дуже корисно для розробника і робить даний інструмент дуже гнучним. Також часто в розробники стикаються з такими проблемами, що PostgreSQL не може задовільнити жодною частиною свого функціоналу, і тому існує можливість розробити спеціальний плагін, щоб покращити його, наприклад написавши власний оптимізатор [22].

Якщо виникають якісь питання потрібно пам’ятати що спільнота даного продукту велика і багато розробників по всьому світу з радістю допоможуть вам з вашими помилками.

2.9. HBase

Apache HBase дану базу даних відносять до нереляційних баз даних, її код є відкритим та написана вона з використанням мови Java. Дана СУБД відноситься до сімейства стовпців, тобто це колонково-орієнтоване сховище типу ключ-значення. Вона працює з системою HDFS і забезпечує можливості BigTable для Hadoop.

HBase забезпечує такий спосіб зберігання наборів даних, які часто зустрічаються і повторюються. Дану СУБД часто використовують для обробки даних у режимі реального часу. Також HBase надає можливість випадковий доступ для читання чи запису великих обсягів даних. HBase не підтримує мову запитів, як SQL.

Модель характеризується такими принципами:

- Всі дані зберігаються в таблиці, вона є проіндексована первинним ключем.
- Первинний ключ може зберігатися як необмежений набір атрибутів.
- Колонки формуються в групи колонок, також список і назви цих груп фіксований і має чітку схему.
- Записи зберігаються в відсортованому порядку за первинним ключем.
- Отсортований список формують ті атрибути, які притаманні одній групі колонок і відповідні одному ключу.

Переваги використання Apache Hbase:

- Може мати необмежену кількість стовпців, дані зберігаються по стовпцями і тому немає необхідності зберігати значення. В цьому випадку HBase добре підходить для розріджених наборів даних.
- Забезпечує високу швидкість роботи при будь-якому масштабі. HBase спроектована для підвищення продуктивності для систем в яких вузли постійно зростають, де рядки рахуються мільйонами, загальне представлення виконавчої архітектури зображено на рис. 2.8.

- Надає великий набір API-функцій, таких як: видалення значення атрибута будь-якого ключа, видалення цілого стовпця, видалення всього сімейства стовпців. Також ще є набір функцій, як: Put, Get і Delete тощо.
- Підтримка лексикографічного впорядкування даних по ключам рядків. Ключі в рядках завжди відсортовані і це слугує аналогом індексу первинного ключа, який застосовується в реляційних БД.

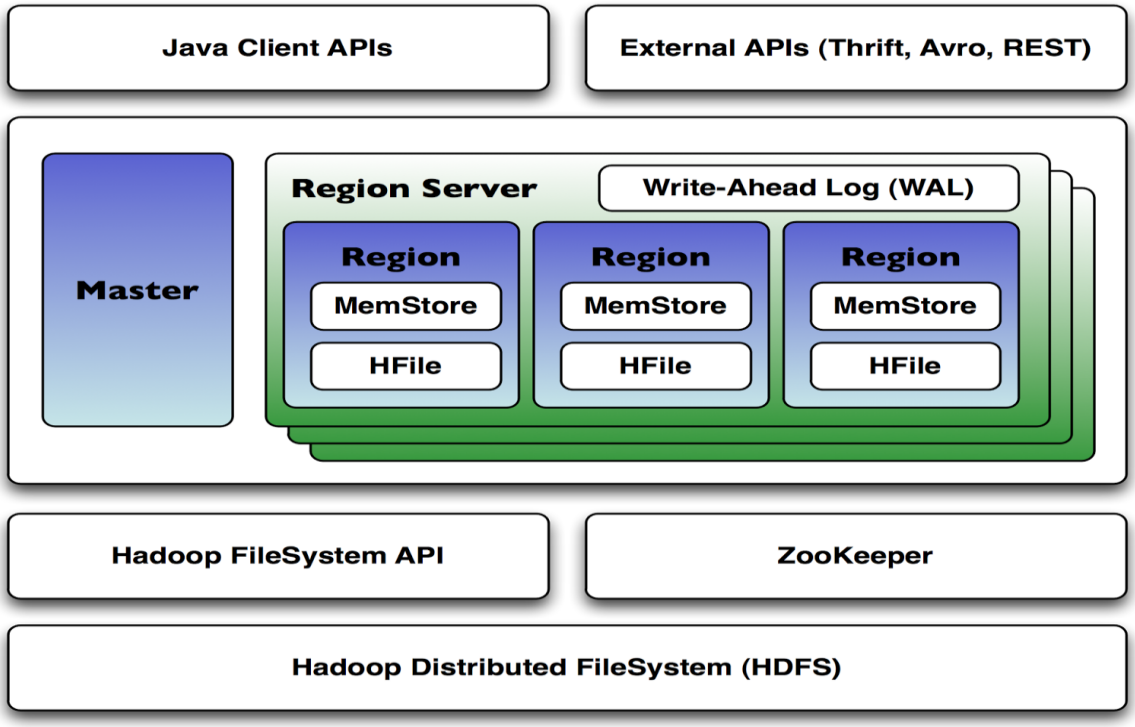


Рис. 2.8. Модель архітектурного підходу в Apache Hbase [23]

ВИСНОВКИ ДО РОЗДІЛУ 2

В цьому розділі піднято та проаналізовано питання використання сучасних технологій, які є популярними сьогодні на ринку і використовуються для розробки веб-додатків. Всі розглянуті технології дуже зарекомендували себе та широко використуються розробниками по-всьому світу. Кожна з розглянутих технологій в чіткій мірі покривала вирішення проблеми написання своєї частини в клієнт серверній архітектурі. Був проведений аналіз та дослідження таких тенологій як: Java Spring, NodeJs, Python Django – для серверної частини, ReactJs, AngularJs, VueJs – для клієнтської частини та PostgreSQL, MongoDB – в ролі баз даних.

Коли мова йде про серверну частину, то всі з запропонованих інстументів є дуже потужними. Кожен з даних фреймворків має як свої плюси так і мінуси. Мій вибір зупинився на Java Spring Framework. Чому саме він:

- Є практичний досвід з використання даного фреймворку
- До складу Spring Framework входить багато інших підпроектів, які надають вже певні готові рішення, такі як: SpringMVC, Spring Security, SpringData тощо
- Присутність інверсії управіння
- Присутність анотацій
- Хороша офіційна документація
- Підтримка та інтеграція з технологіями доступу до даних

Перейдемо для вибору технологій які зв’язані з клієнтською частиною. Особисто в мене не було великого досвіду з написанням клієнтської частини з такими технологіями, але мій вибір керувався тим, яка все ж таки технологія широко використовується на ринку і тому після переглядів вакансій та повторному аналізу я обрав AngularJs. Особисто для себе виділив такі переваги як:

- Модульність та використання спеціалізованих деректив

- Велике ком’юніті, яке в разі проблеми може допомогти
- Декларативний стиль написання коду
- Корисний функціонал для SPA
- Двухстороння зв’язаність даних тобто зміни в користувацькому інтерфейсі відразу відображаються на об’єктах нашого додакту і навпаки

Перейдемо до вибору бази даних вибір був між класичним підходом об’єктно-реляційним PostgreSQL та документоорієнтованою MongoDB. Мій вибір впав на класичний підхід з використанням PostgreSQL, адже я не мав ніяких складних за структурою даних.Також особисто можу відмітити такі переваги:

- Структури і типи даних, є можливість створення власного
- Розмір даних
- Об’єктно-орієнтована СУБД
- Цілісність даних, він є захищеним і відповідає всім принципам ACID

Отже після проведеного аналізу в технологіях прийшов до висновку, що в побудові програмного додатку буде використовуватись такі технології як: Java Spring Framework, AngularJs, PostgreSQL.

В наступних розділах піде мова про проектування веб-сервісу тайм-трекінгу, який буде використовувати даний стек технологій, який зазначений вище.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ ТАЙМ-ТРЕКІНГУ

3.1. Опис завдання

Для реалізації ідеї побудови системи тайм-трекінгу було обрано такий стек технологій як: Java Spring Framework, AngularJs та PostgreSQL, адже метою створення даного програмного забезпечення була оптимізація керування роботою над проектами та завданнями, а також надання можливості користувачу вести власну статистику затраченого часу на виконання завданнь. В свою чергу даний система надає користувачу таку функціональність як:

- Можливість аунтентифікації та авторизації для користувача
- Сторення нового проекту
- Можливість назначити відповідального за виконання проекта
- Пошук по назві проекта
- Видалення проекту
- Створення завдань до проекта
- Пошук по назві завдання
- Видалення завдання
- Можливість назначити відповідального за виконання завдання
- Можливість зміни ролі
- Можливість ведення затраченого часу на виконане завдання, тобто ведення особистої статистики

Як вже згадувалось в попередньому розділі, концепцією для розробки даного веб-додатку послугувала клієнт серверна архітектура, зображення даної концепції зображено на рис. 3.1. Дана архітектура фундаментально складається з таких компонентів, які мають велике значення в розробці даного продукту і в цілому при розробці веб-додатків також, до них відносяться:

- ✓ В свою чергу клієнт звертається до сервера, для отримання певної інформації
- ✓ Сервери по запиту надають дану інформацію
- ✓ Ну і до компоненту також можна віднести мережу, яка конролює взаємодію цих двох компонентів

Даний архітектурний підхід, означає, що існує тісний перерозподіл обов'язків між самим сервером та клієнтом, виділяють такі обов'язки як:

1. Рівень клієнта виступає в ролі рівня по відображенню даних, тобто це звичний для всіх інтерфейс за допомогою якого відбувається візуалізоване надходження інформації.
2. Серверний рівень або як його ще називають прикладний, даний рівень несе головну логіку по обробці та наданню інформації.
3. Рівень сервер даних або БД виступає в ролі рівня який відповідає за керування даними чи то зберігання, оновлення, видалення тощо.

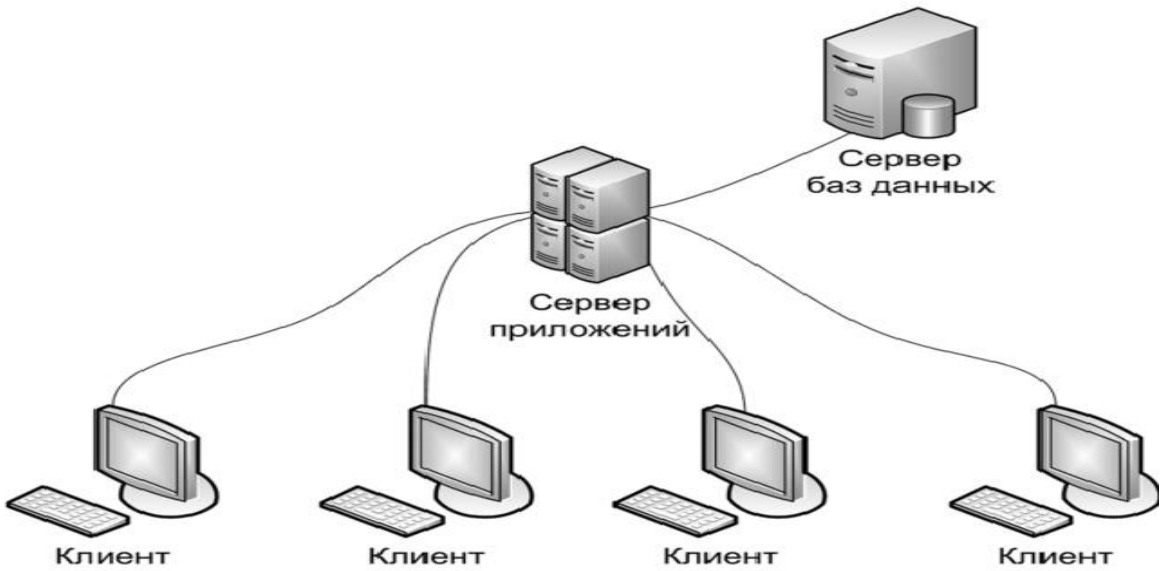


Рис.3.1. Схема клієнт серверної архітектури

3.2. Структура MVC та її аналіз

Сучасні веб-сервіси мають динамічний функціонал та дружній інтерфейс і тому при проектуванні даної системи було вирішено використовувати такий архітектурний шаблон або як ще його називають патерн проектування MVC. Головною причиною для використання MVC є розділення таких понять, як: дані та бізнес-логіка від візуалізації. Ця проблема виникає тому, що звичайна статична сторінка, яка написана на HTML не може реагувати на різноманітні маніпуляції з боку користувача. Для забезпечення цієї взаємодії потрібні динамічні веб-сторінки, в свою чергу MVC і є ключем вирішення цієї проблеми і тому більшість веб-додатків будуються за допомогою використання MVC.

MVC або Model-View-Controller розшифровується взагалі, як модель-представлення-контролер. Даний спосіб заключається в розбитті на блоки, які будуть вирішувати і нести певну логіку програмного продукту: один блок відповідає за дані, тобто модель, другий - за зовнішній вигляд, а третій контролює цю взаємодію, схема даної взаємодії зображена на рис. 3.2. Розглянемо компоненти більш детально:

- Model - цей компонент шаблону є центральним відповідає за дані, а також в певній мірі визначає структуру програми.
- View - відповідає за взаємодію з користувачем, тобто за представлення інформації і способи її використання.
- Controller - даний компонент реагує на дії користувача і допомагає інтерпретувати ці дії у команди для моделі або навпаки від моделі у view.

На сьогодні багато веб-фреймворків побудовані на парадигмі MVC. Тому у вас не буде виникати проблем, щоб зрозуміти принцип іншого фреймворку, в даному випадку Java Spring Framework надає можливість використовувати Spring MVC для більшої абстракції додатків.

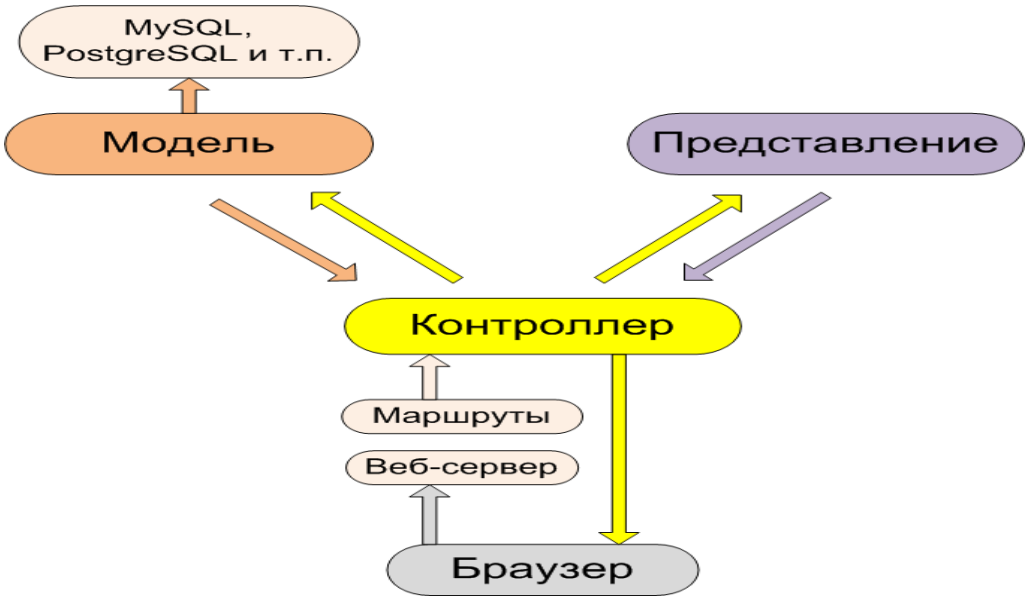


Рис.3.2. Схема роботи MVC

Розглянемо переваги використання MVC:

- Єдина концепція системи, яка забезпечує глобальну архітектуру веб-додатку.
- Блоки model, view та controller – є доволі незалежними одне від одного і тому при зміні чогось одного ми не зруйнуємо інше.
- Даний принцип допомагає зробити наш код більш розширюваним, щоб він підпорядковувався принципам SOLID.
- Доступна можливість використання декількох View для однієї Model.

Розглянемо недоліки використання MVC:

- Так як дані три блоки є майже незалежними, то всі взаємодії відбуваються через передачу даних і це призводить до використання великої кількості ресурсів.
- Так як кожен функціональний модуль має притримуватись розбиття на три модуля це в деякій мірі ускладнює розбиття архітектури на модулі.
- З проблеми вище випливає наступна проблема пов’язана з процесом написання нового функціоналу та поєднанням з вже існуючим функціоналом [24].

3.3. Покрокова розробка даного додатку

Після проведення дослідження та вибору технологій розробки, що було проведено в 2 розділі, дана система буде реалізована у вигляді веб-додатку її основні частини: клієнтська частина, серверна та база даних. Розглянемо більш детально серверну частину, вона містить в собі такі частини як: Model, Repository, Service, Controller та саму базу даних, така взаємодія схематично зображена на рис. 3.3.

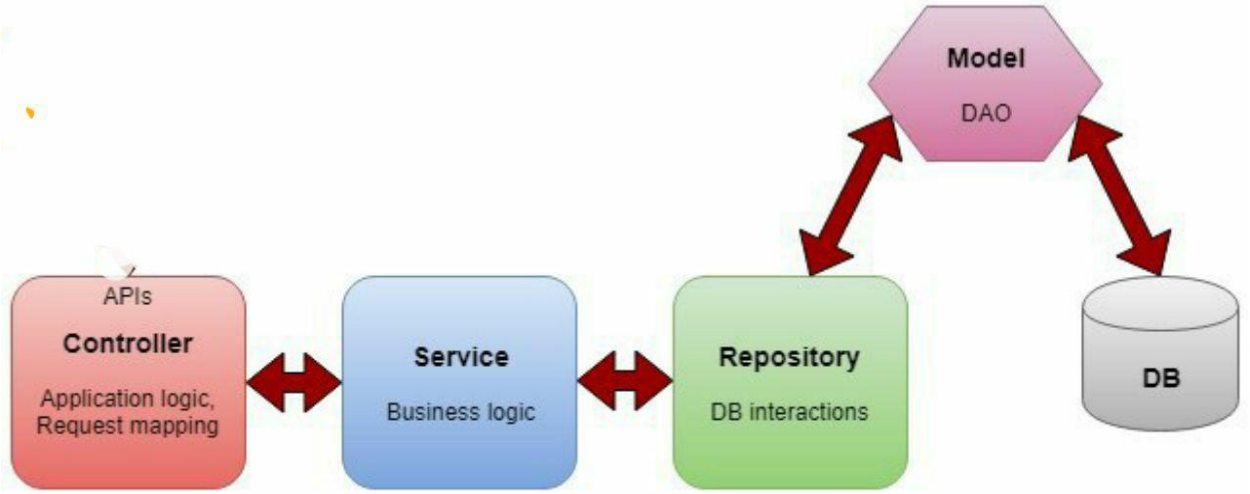


Рис. 3.3. Схема взаємодії серверної частини

Першим кроком нам потрібно спроектувати базу даних в якій будуть зберігатись наші дані. Враховуючи всі вимоги до продукту, який розробляємо спроектуємо таку схему бази даних даної системи. Дані таблиці опис яких буде нижче, будуть повністю співпадати з Model. Адже Model – це суто представлення об’єктів в програмному коді, яке рахується за реальну сутність з своїми полями, а БД повинно зберігати кожне вказане поле.

Перша таблиця має назву User і вона зберігає всю інформацію про користувачів, вона складається з таких полів як:

- user_id – унікальний індентифікатор кристувача
- creating_data – дата створення користувача
- editing_date – дата змінення профілю
- latest_activity – дата останньої активності
- account_enabled – поле, яке означає, що акаунт доступний в системі
- credentials_non_expired – перевірки повноважень при вході

- first_name – ім’я користувача, який є в системі
- last_name – прізвище користувача
- user_email – пошта людини, яка використовує даний сервіс
- user_password – особистий пароль для входу в дану систему
- user_name – нік-нейм для системи
- user_role_id – поле для поєднання з таблицею ролей

Наступна таблиця, яка необхідна для коректної роботи таблиці описаної вище - це таблиця User_Roles, в свою чергу вона складається з таких полів:

- user_role_id - унікальний індентифікатор ролі
- creating_date – дата створення
- editing_date - дата останньої зміни
- user_role_name – назва ролі необхідна для коректної роботи системи

Між таблицями User та User_Roles існує відношення Many-to-One, це означає що користувач може мати тік одну роль, а ролі можуть відноситись до багатьої юзерів.

Головною таблицею при проектуванні даної системи виступає таблиця Projects, для коректної роботи вона містить такі поля:

- project_id - індентифікатор проекту, визначає його унікальність
- creating_date – дата створення проекта
- editing_date – дата останніх коректних змін в проекті
- project_name – назва проекта, який розробляють
- user_owner_id – унікальний індентифікатор користувача для того, щоб назначити відповідального за виконання даного проекту

Між даною таблецею та User відношення також Many-to-One.

Таблиця Projects_Executors дана таблиця виступає, як асоціативна між таблицями Project та User, в ній є такі поля:

- project_executor_id – індентифікатор даної таблиці
- user_id – ідентифікатор відповідного працівника
- project_id – ідентифікатор проекта з яким зв’язаний працівник

Таблиця Tasks використовується для збереження завдань, які прив’язані і містять в кожному проекті. І тому побуло прийнято рішення будувати таблицю з такими полями:

- task_id - унікальний індентифікатор для кожного завдання
- creating_date – дата створення завдання
- editing_date – дата останньої зміни самого завдання
- task_name – назва завдання
- user_owner_id – індентифікатор працівника, який відповідає за виконання даного завдання
- project_id - ідентифікатор проекта, якому належить саме завдання

За аналогією до Projects_Executors таблиця Task_ Executors є асоціативною таблицею між таблицями User та Tasks і складається з таких полів:

- task_executor_id – індентифікатор даної таблиці
- user_id – унікалькальний індентифікатор працівника, для прив’язки з завданням
- task_is – унікальний індентифікатор завдання для того щоб призначити відповідного для виконання завдання

Для ведення трекінку часу на виконання завдання створимо таблицю Tracks, яка буде складатись з таких полів:

- track_id – ідентифікатор для даної таблиці
- creating_date – дата створення запису в таблицю трекінгу
- editing_date – дата внутрішніх змів цієї таблиці
- date_activity – дата активності
- track_name – назва даного трекінгу
- time_value – затрачений час на виконання даного завдання
- task_id – ідентифікатор таски за яким закріплений трекінг по часу
- user_owner_id – ідентифікатор користувача якому належить даний трекінг

Розглянемо, як працює Service та Repository. Внутрішня структура серверної частини зображена на рис. 3.4. В Service знаходиться майже вся бізнес-логіка додатку і для того, щоб якось працювати з даними вона звертається до Repository. Репозиторій в свою чергу забезпечує доступ до даних через використання драйверів, які з'єднуються з базою даних, а також забезпечує виконання операцій з даними. При розробці даного програмного додатку було використано такі сервіси та репозиторії:

ProjectRepository – даний репозиторій використовується для того, щоб забезпечити доступ до даних, які пов'язані з проектом та виконання CRUD-операцій, які необхідні для коректної роботи.

TaskRepository – цей репозиторій спроектований з метою, надання доступу до даних, які логічно пов'язані з завданнями. Також репозиторій надає можливість використовувати операції з даними, для підтримки роботи.

TrackRepository – забезпечує доступ до інформації пов'язаною з веденням статистики виконання завдань по проекту.

UserRepository – надає доступ до інформації про користувача даної системи.

UserRoleRepository – створює можливість роботи з даними, які тісно пов'язані з ролями в системі.

UserService – цей сервіс забезпечує виконання бізнес-логіки зв'язаної з користувачом,а саме: авторизаці та аутентифікації користувача, можливість пошуку по імені, редагування та демонстрація інформаці про користувача.

UserRoleService – сервіс надає можливість працювати з ролями, які являються бізнес одиницею даного сервісу.

TrackService – забезпечує виконання всіх операцій для відслідковування часу, затраченого розробником для виконння завання, це такі операції: видалення, створення, редагування та демонстрація.

TaskService – він має такий функціонал, який надає можливість менеджеру створювати завдання і вести повноцінну роботу з завданнями, які прив'язані до проекту.

ProjectService – має аналогічний функціонал як і TaskService, тобто він відповідає повністю за всю необхідну бізнес-логіку, яка пов’язана з проектами.

Кожен з даних сервісів несе в собі бізнес-логіку,яка забезпечує валідацію даних для коректної роботи програмного забепечення. В свою чергу контролери забезпечують роботу і тісну взаємодію між Model і View.

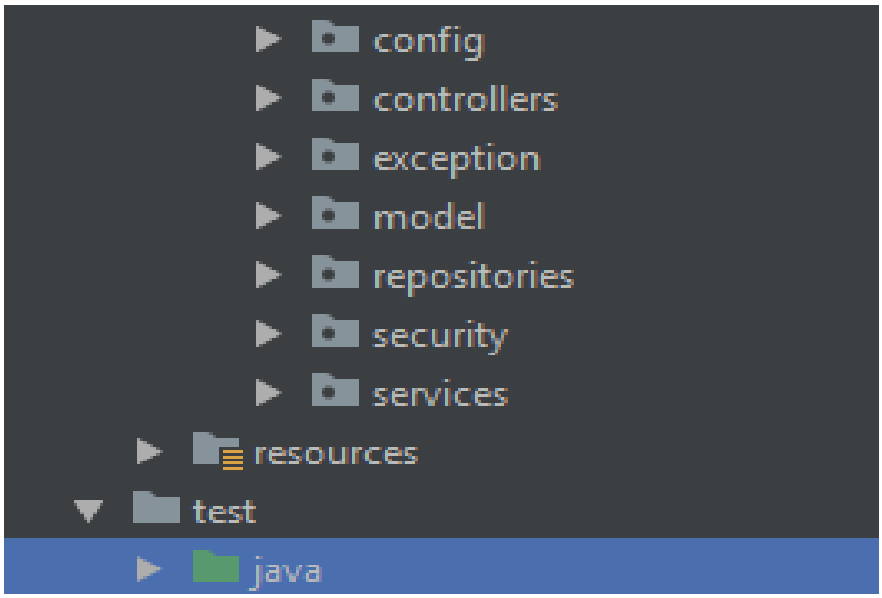


Рис. 3.4. Структурна схема даного програмного забезпечення

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному, третьому розділі, було описано основні компоненти програми та їх взаємодію. Під час створення даного програмного забезпечення була чітко створена клієнт-серверна система, яка має підтримку, інтеграцію з PostgreSQL.

Архітектурний підхід, який використовувався для побудови серверної частини, а саме MVC, забезпечив чітке логічне розбиття компонентів веб-застосунку, які зрозумілі розробнику. Для стилізації та візуалізації даних використовувався AngularJs разом з необхідними бібліотеками. Взаємодію цих двох компонентів було протестовано для забезпечення коректної роботи, тестування проводилось згідно з усіма правилами, для того, щоб даний продукт був конкуренто-спроможній і при подальшому розвитку даного продукту перевіряти роботу написаного коду.

Всі поставлені завдання були реалізовані і вони відповідають вимогам, що були зазначені в технічному завданні даної роботи. Кожна з користувацьких ролей, які присутні в даному веб-застосунку чітко виконує свою бізнес-логіку. Для того щоб даний продукт мав належний рівень захисту використовувалось Spring Security, яке допомогло створити чіткий та захищений механізм авторизації та аутентифікації.

Система має можливість розширюватись, адже вона проектувалась за сучасними архітектурними патернами.

РОЗДІЛ 4
РОЗГЛЯД СТВОРЕНОГО ТАЙМ-ТРЕКЕРУ

4.1. Демонстрація роботоспроможності програми

Результатом написання даної бакалаврської роботи є веб-додаток, який вирішує проблеми менеджменту над проектами та ведення статистики затраченого часу. Тому в цьому розділі мова піде про створений продукт, а саме: ілюстративно продемонструємо роботу програми, ілюстрації в свою чергу виступають як інструкція по використанню даного додатку. Головною частиною кожного продукту є користувацький інтерфейс. І саме він має бути максимально зрозумілим для людей. Адже, людина, яка вперше відкриє даний додаток, якщо не зможе зрозуміти, як ним користуватися, то понатискавши на різні кнопки, вона розчарується і покине ресурс, можливо, назавжди. Для запуску даного програмного забезпечення слід обрати будь-яку операційну систему та встановити і налаштувати: JDK, Intellij IDEA, Node, PostgreSQL, після компіляції запустити програмне забезпечення. Після того як основні кроки виконані відкриваємо браузер і попадаємо на головне вікно даного застосунку, яке зображено на рис. 4.1.

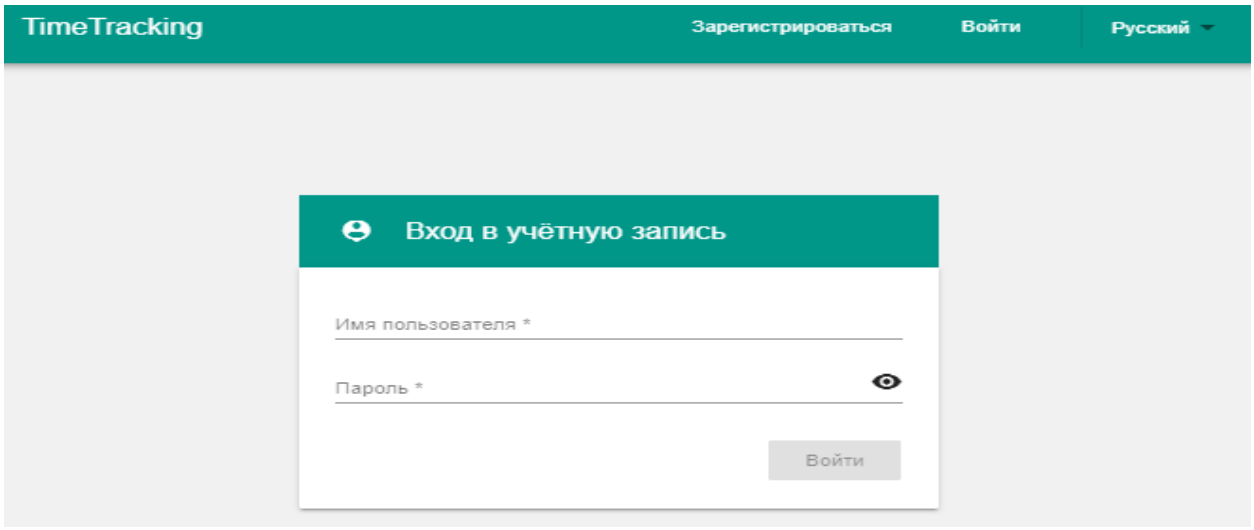


Рис. 4.1. Головний екран даного застосунку

Щоб пройти реєстрацію в даній системі, потрібно в верхній панелі натиснути на кнопку “Зареєструватися”. Результат - реєстраційна форму, яка зображена на рис. 4.2.

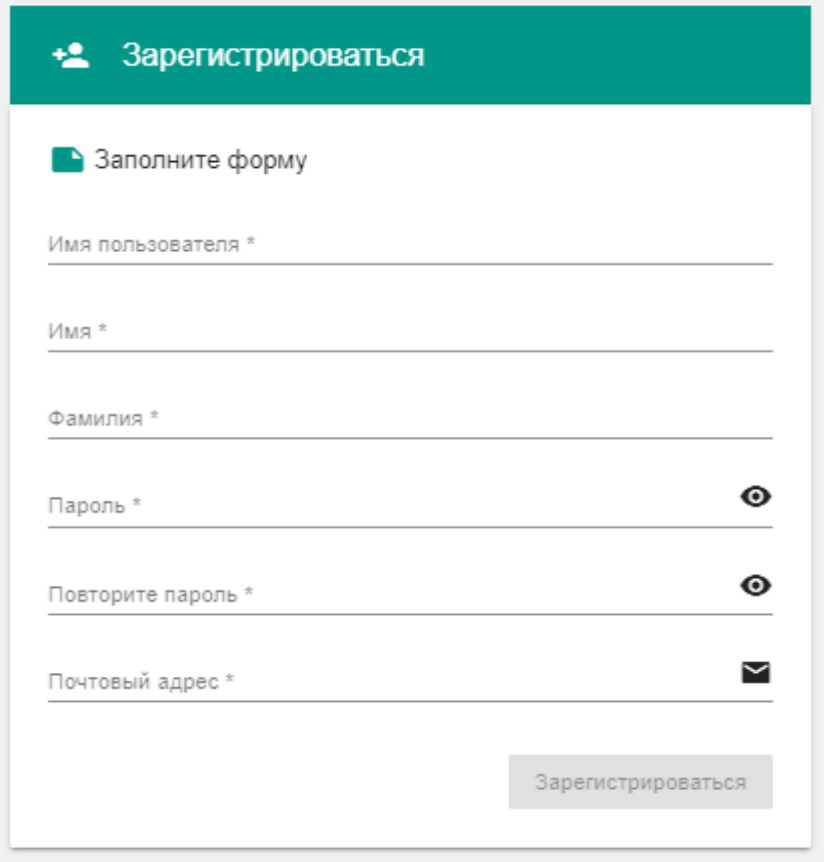


Рис. 4.2. Форма для реєстрації в даному веб-застосунку

Також цей веб-застосунок відповідає всім нормам безпеки і тому програмний додаток передбачає, що пароль буде скритий і сторонні особи не зможуть побачити пароль. Щоб переконатись в тому, що пароль написаний коректно, можна просто нажати на бічний символ, як це зображено на рис. 4.3. В подальшому пароль шифрується і зберігається в базі даних. Всі поля проходять валідацію як з боку клієнтської частини так і з серверної, це продемонстровано на рис. 4.4.

Рис. 4.3. Можливість скрити пароль від сторонніх осіб

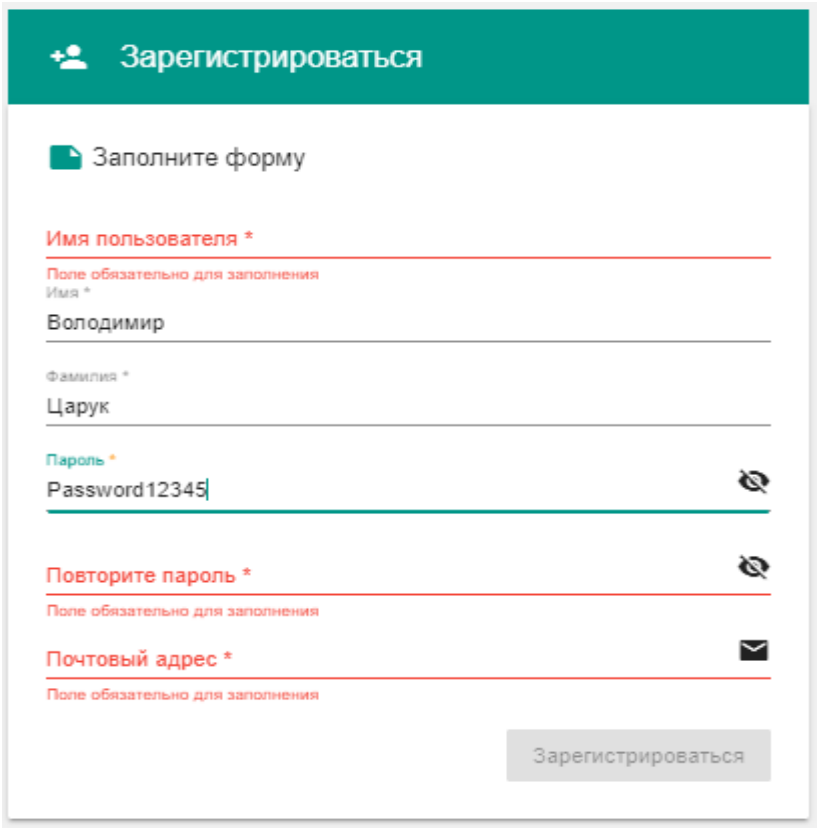


Рис. 4.4. Демонстрація валідації на клієнтській частині

Після проходження реєстрації в верхньому меню обераємо кнопку “Увійти” і вводимо данні для входу це зображено на рис. 4.5.

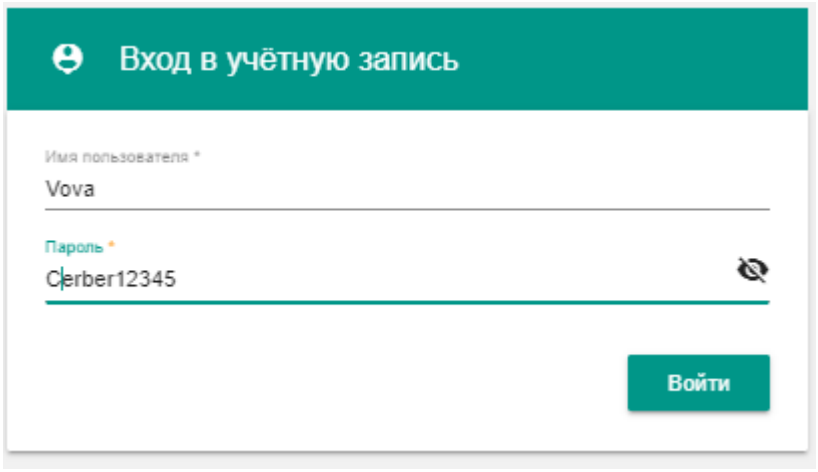


Рис. 4.5. Вхід в обліковий запис

Натиснувши кнопку “Увійти”, ми потрапляємо в головне меню нашого додатка, а саме в профіль користувача, де він може переглянути список трекінг свого часу на завтрачені завдання та самі завдання на які він затверджений і несе відповідальність за їх виконання. З ліва зображено допоміжне меню з пунктами профіль та проекти, це зображено на рис. 4.6.

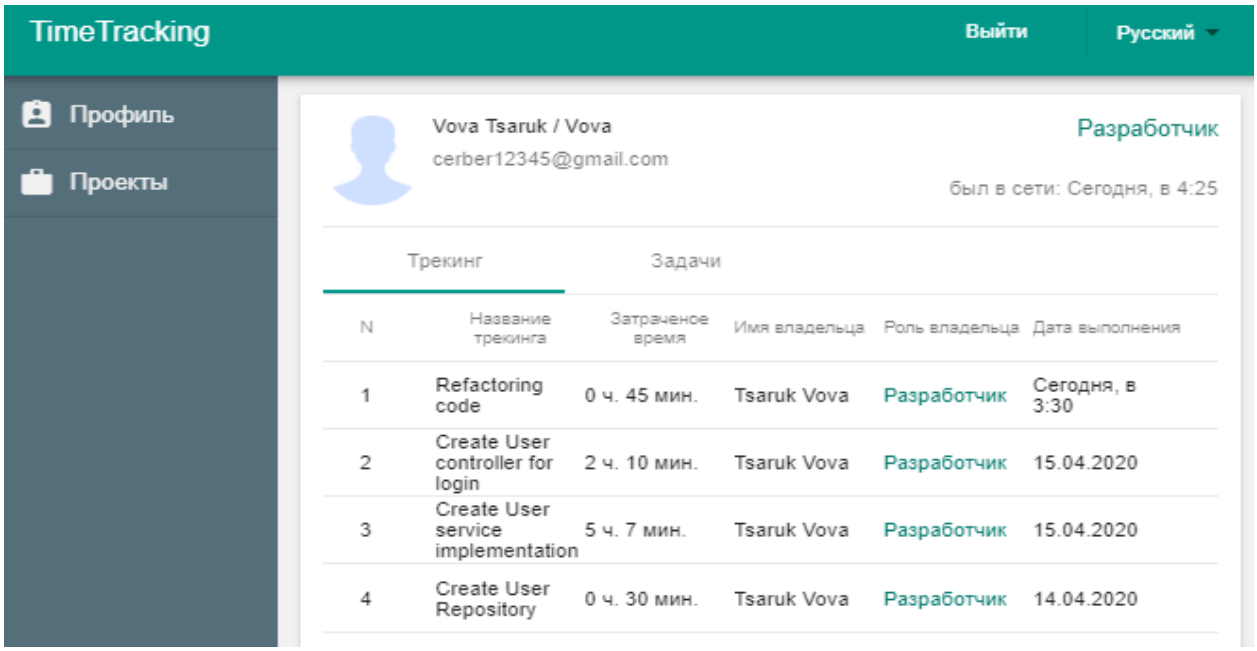


Рис. 4.6. Головна сторінка розробника

Якщо натиснути кнопку “Проекти”, то ми побачимо усі доступні проекти, які зараз розроблюються, приклад такого зображений на рис. 4.7. Можна злегкістю вести пошук по назві проекта та сортувати як по назві так і по діті створення.

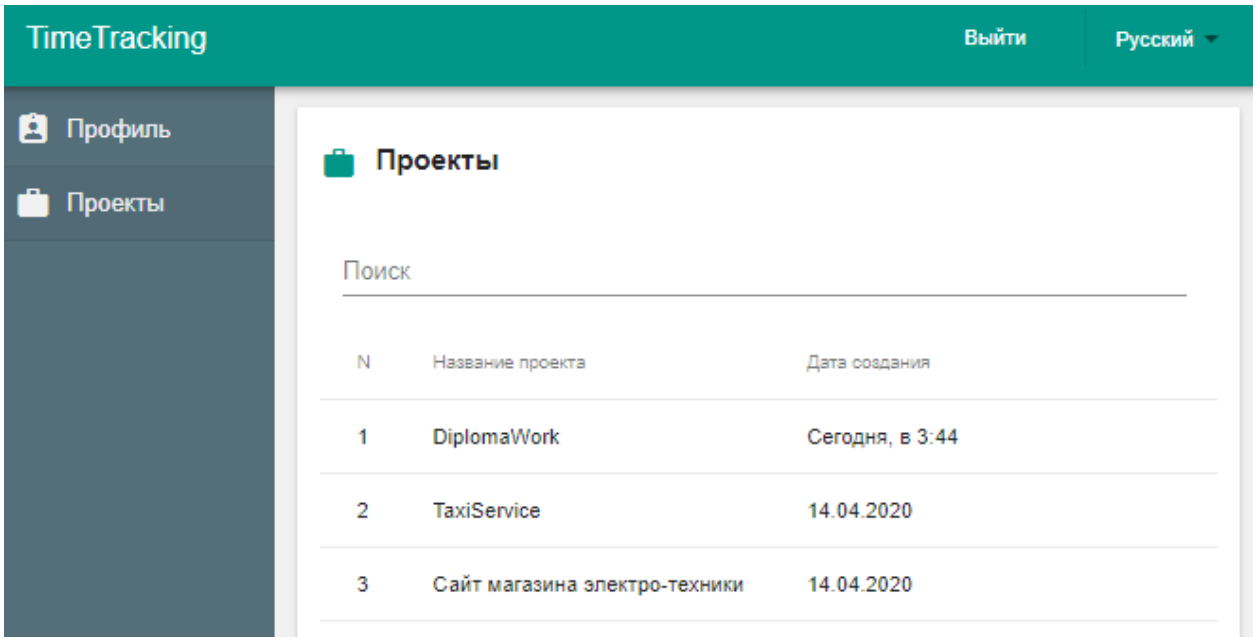


Рис. 4.7. Демонстрація проектів над якими зараз працює компанія

Для перегляду списку завдань, які необхідно зробити для даного проекту достатньо просто натиснути на назву проекта, результат цього зображений на рис. 4.8.

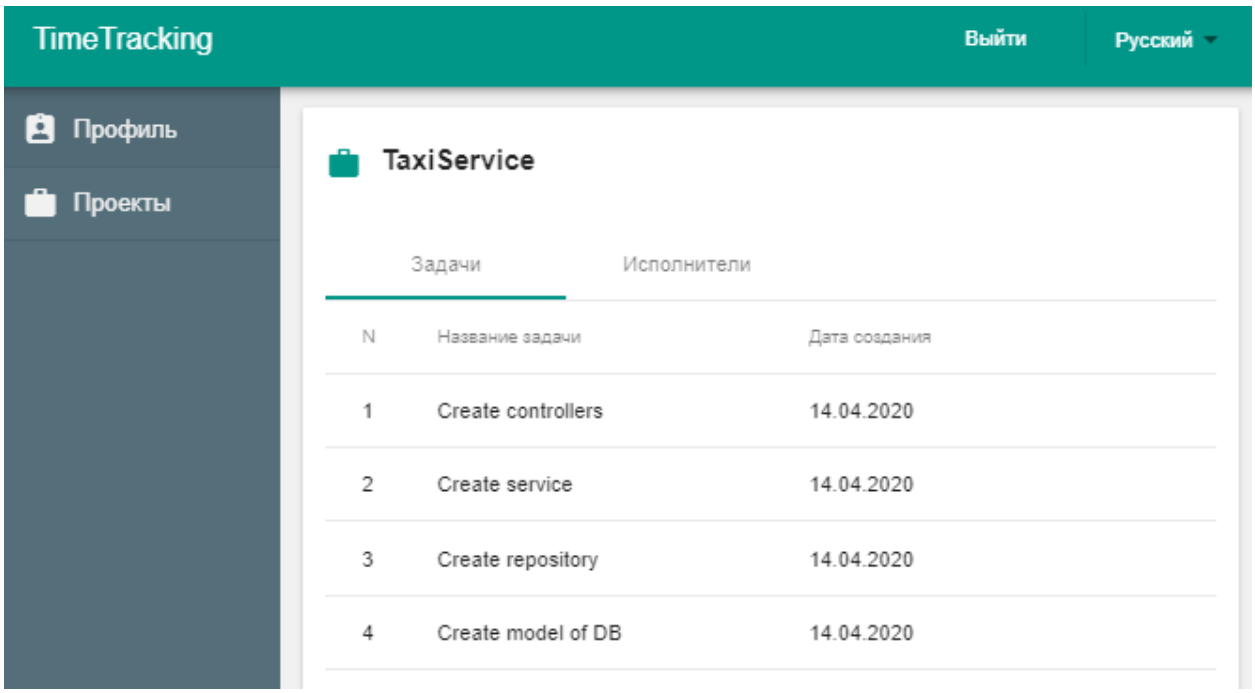


Рис. 4.8. Список завдань, які характерні для проекта ”TaxiService”

Менеджер затверджує команду розробників для виконання завдань. Розробник, який затверджений на дане завдання, може відзначати етапи розробки даного завдання, тобто якщо говорити іншими словами вести трекінг часу на певні етапи. Для цього йому буде достатньо натиснути кнопку “Добавити трекінг” і після цього він отримає часову форму для заповнення, як це зображено на рис. 4.9

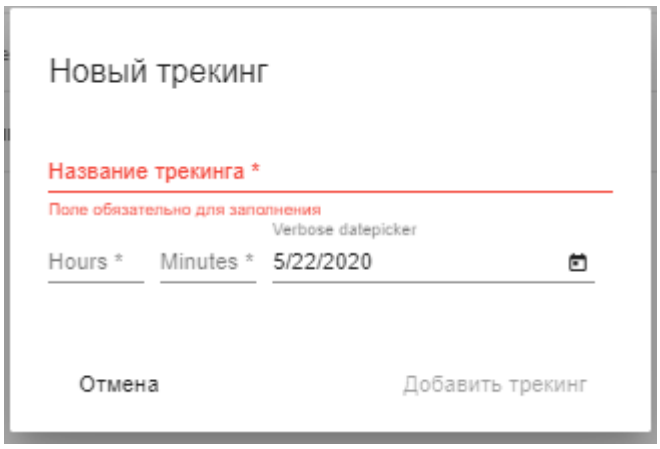


Рис. 4.9. Форма для ведення часового трекінгу

Після підтвердження додавання трекінгу часу ми отримуємо ось такий результат, який зображений на рис. 4.10. Також кожен з розробників даного проекту може переглянути, хто відповідальний за дане завдання.

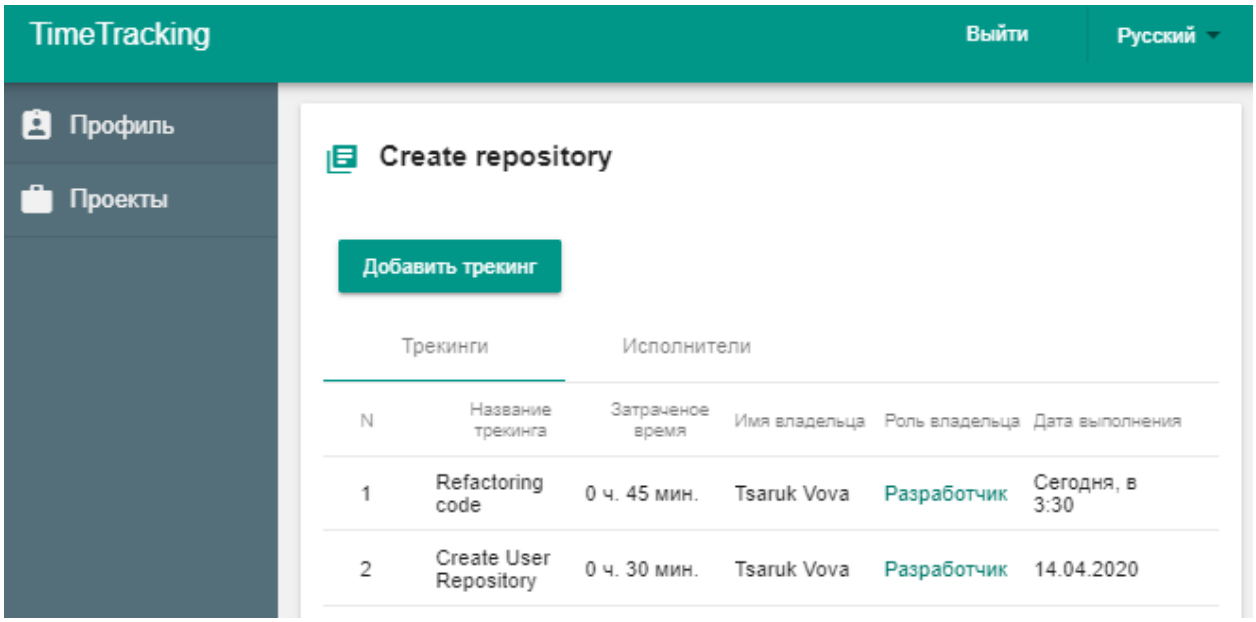


Рис. 4.10. Фіксування виконання завдань

В даній системі, окрім ролі розробника, яка надається при реєстрації, є ще ролі менеджера та адміна. Менеджер це одна з центральний ролей, яка керує життєвим циклом проектів, завдань, затвердженням команди і тому подібних речей. В його повноваження входить створення нових проектів та завдань. Якщо натиснути на нопку створити проект, ми отримуємо таке діалогове вікно в якому також передбачена валідація, як і в усіх діалогових вікнах даного сервісу, які потребують введення тексту. В даному випадку нам потрібно задати назву проекту, приклад цього на рис. 4.11.

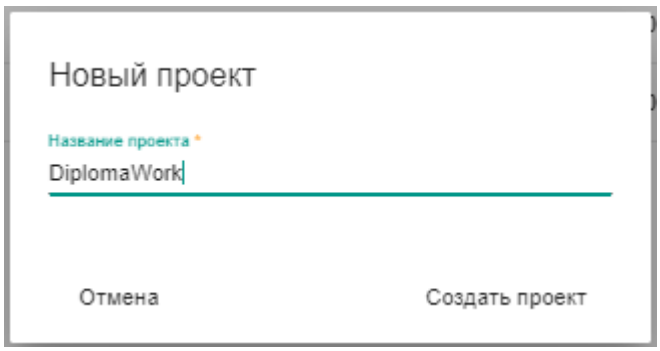


Рис. 4.11. Створення нового проекту

Після створення даного проекту, ми переходимо на нього, для цього нам достатньо натиснути на його назву. Після цього нас зустрічає такий перелік можливостей як: створити завдання, добавити виконавця, видалити виконавця, приклад даного меню зображений на рис. 4.12.

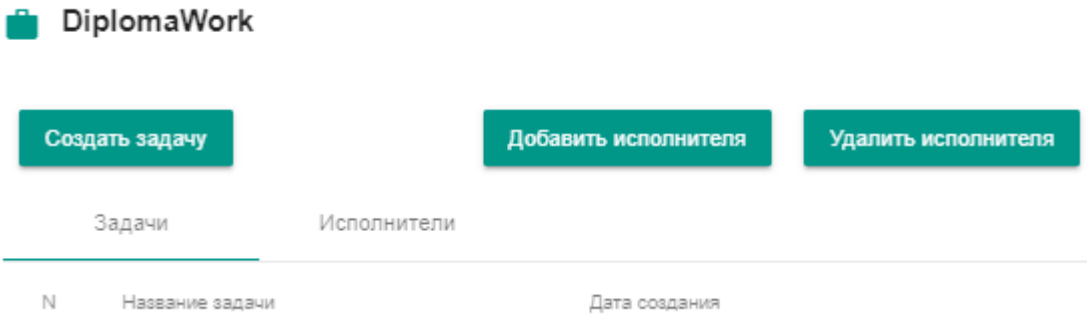


Рис. 4.12. Меню менеджера по керуванню життєвим циклом проекта

Для створення нового завдання до даного проекта нажимаємо на кнопку “Створити завдання”, отримаємо діалогове вікно, яке зображено на рис. 4.13. Для підтвердження створення завдання достатньо натиснути на відповідну кнопку.

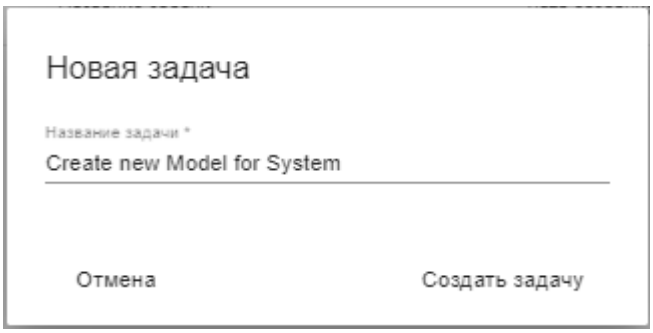


Рис. 4.13. Демонстрацій створення нового завдання

Після цього менеджер повинен обрати команду для розробки даного проекту для цього потрібно натиснути кнопку “Добавити виконавця” і обрати команду з людей, які зареєстровані в системі, як це зображено на рис. 4.14.

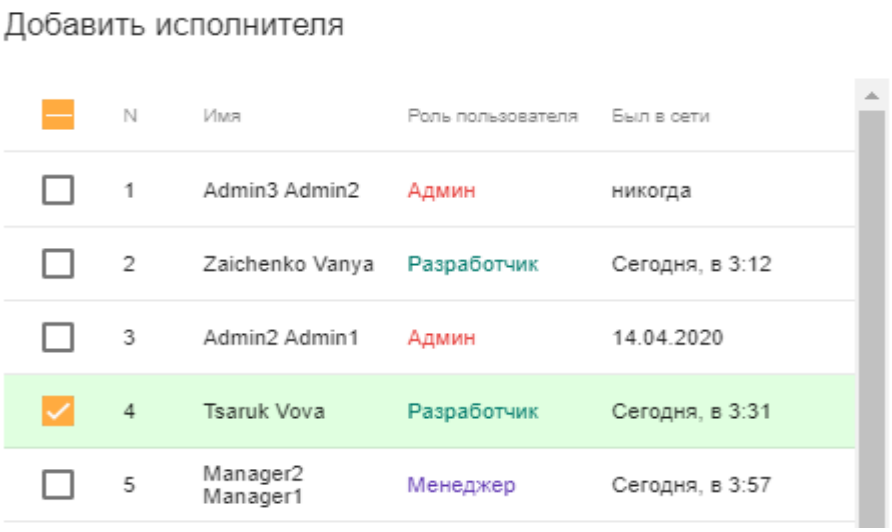


Рис. 4.13. Затвердження команди на проект менеджером

Аналогічно цьому відбувається видалення людей з проекту. Також в менеджера є можливість назначити конкретного розробника на виконання завдання та також його видалення, така можливість продемонстрована на рис. 4.14.

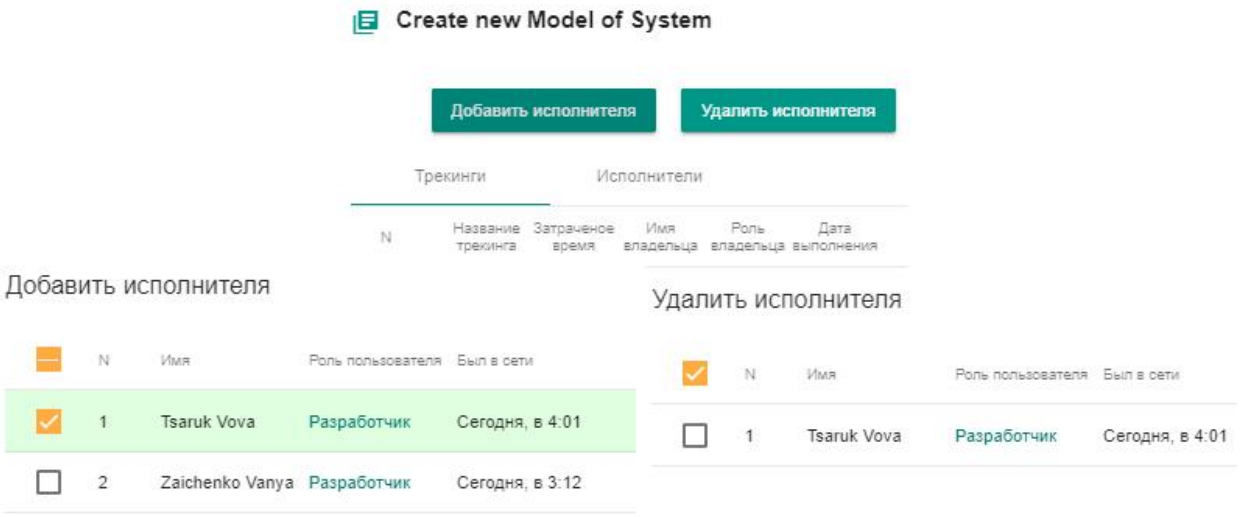


Рис. 4.14. Затвердження та видалення розробника на виконання завдання

Якщо говорити за роль адміна, то йому доступний аналогічний функціонал, що і менеджеру, а також він може змінювати ролі користувачів в системі, як це зображено на рис. 4.15.

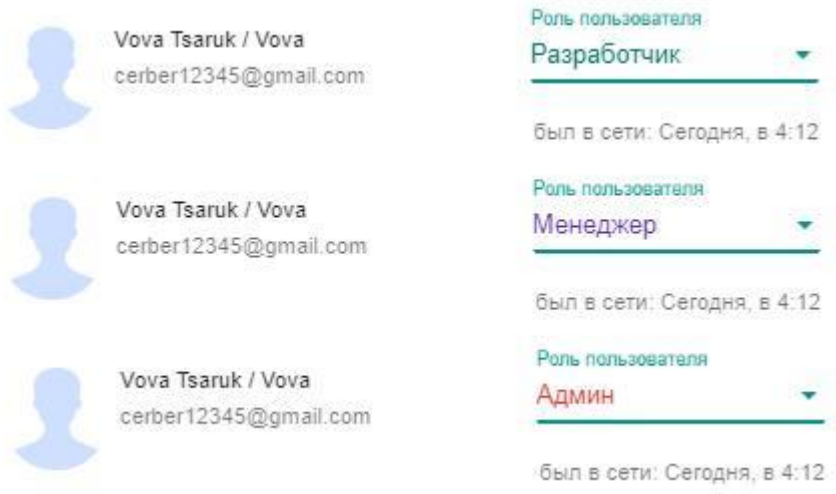


Рис. 4.14. Зміна адміном ролей системи

4.2. Тестування тайм-трекера

Одним з етапів розробки програмного продукту є тестування. Воно спрямоване на перевірку коректної роботи продукту, пошук помилок та їх швидке виправлення. Першим етапом, якщо говорити за тестування, то були

написані автоматичні юніт тести для перевірки коректної роботи створеного функціоналу. Адже чим більше стає програмний додаток тим важче слідкувати чи коректно працює раніше написаний функціонал.

Після закінчення написання автоматичного тестування, наступним етапом було димове тестування. Цей метод говорить про те, що була сформована група з 10 людей, які використовували основний функціонал. У процесі такого тестування були виявлено декілька граматичних помилок та проблему з валідацією даних на серверній частині. Також після такого методу було отримано відгуки про написаний користувацький інтерфейс. Усі відгуки були ретельно проаналізовані та прийняті до уваги.

Тестування даного веб-трекера проводилося на кожній ітерації створення.

4.3. Виділення основних переваг в порівнянні з існуючими аналогами

Під час проектування даного веб-сервісу велося детальне дослідження аналогічних систем, мова про які велась в розділі 1. Всі перелічені аналоги, які згадувались в даному розділі мають як свої переваги, так і недоліки.

При написані системи тайм-трекінгу було дотримано всіх умов, які задані в технічному завданні, а саме:

- Наявність особистого кабінету
- Авторизація та аутентифікація користувачів
- Можливість створення проекту
- Пошук по назві проекту
- Додавання завдань до проекту
- Пошук завдань по назві
- Можливість затвердити людей на проект
- Назначити відповідальних за виконання завдання
- Можливість вести особисте відслідковування часу для підвищення продуктивностії
- Зміна локалізації

До основних переваг даного веб-застосунку можна віднести такі речі як:

- Відсутність рекламного контенту
- Дружній інтерфейс
- Чітко підібраний функціонал, який легкий в освоєнні
- Дуже зручний для невеликої команди
- Безкоштовний продукт
- Розширені та логічно згруповані права усіх учасників

4.4. Рекомендації щодо подальшого вдосконалення та розширення

Хочеться виділити таку річ, як можливість загального аналізу роботи в системі. Дана можливість зацікавила б багатьох людей, які використовують подібні системи менеджер-трекери. Ідея полягає в тому, щоб створити додатковий додаток для робочого столу, який буде збирати всю інформацію про активні вікна системи і заміряти час проведення на певному сайті, використання певного додатку тощо. Дана статистика повинна відображатись в веб версії, також там повинні бути присутні діаграми, на яких би відображалась інформація про тип проведеного часу. Тобто при використанні гри час фіксувався як проведений на розваги. При написанні коду в спеціалізованому середовищі розробки, час би відображався як затрачений на роботу.

Також цікавою ідеєю було б добавлення додатковий сервісів для прискорення проектної роботи. Наприклад так як багато компаній використовує таку методологію розробки як Scrum, то при проведенні різноманітних демо, ретроспектив і тому подібних речей, коли команда починає оцінювати завдання вони використовують ScrumPoker. Ідея полягає в тому щоб система надавала свої внутрішні подібні сервіси для прискорення розробки програмного продукту.

Так як розробник більшу частину дня проводить в кріслі і веде малорухливий спосіб життя, можна створити додатковий сервіс, який буде надсилати йому рекомендації протягом дня для нагадування про підтримку свого фізичного здоров’я.

ВИСНОВКИ ДО РОЗДІЛУ 4

Підведемо підсумки, в даному розділі було продемонстровано вже створений веб-додаток, застосування даного продукту покриває такі сфери як: тайм-менеджмент, проектний менеджмент, керування життєвим циклом продукту, тощо. Дана демонстрація роботоспроможності створеного продукту, також можна розцінювати, як покрокову інструкцію до даної системи.

Також розглянули питання з тестуванням даного застосунку. При кожній ітерації написання даного продукту проводилось ручне тестування та писались юніт-тести, після цього вносились подальші правки при знаходженні помилок.

При порівнянні з аналогами, мова про які йшла в розділі 1, було виділено основні переваги даного застосунку та розглянуто основний функціонал, який необхідний для коректної підтримки життєвого цикла проекта.

Одним з важливих питань, які піднімались в даному розділі це було питання подальшого вдосконалення та розширення, адже створений продукт, в майбутньому, може вести конкуренцію з аналогами, які зараз присутні на ринку. Запропоновані ідеї для розвитку є цікавими для розгляду і мають право на існування.

ВИСНОВКИ

Результатом написання даної бакалаврської роботи, являється, веб-застосунок для ведення управління проектами та статистики затраченого часу на виконання завдань.

Після проробленої роботи, варто зробити наступні висновки:

1. Було проведено ряд кроків для створення даного програмного забезпечення, а саме: аналіз та дослідження існуючих рішень, проаналізувавши недоліки конкурентів, було знайдено золоту середину в побудові бізнес-логіки додатку та сформовано технічне завдання.

2. Для реалізації продукту було проведено дослідження та аналіз технологій для створення веб-додатків, які зараз використовуються на ринку. Результатом цього була вибрала клієнт-серверна архітектура та такий стек технологій для її реалізації як: для написання серверної частини - мова програмування Java, її фреймворк Spring, для клієнтської частини був використаний AngularJs, та в ролі СУБД було обрано PostgreSQL.

3. Спроектований додаток чітко відповідає прописаним пунктам в техніхному завданні, зокрема, це стосується такого функціоналу як: реалізована авторизація та аутентифікація для забезпечення безпеки даного веб-застосунку, розбиття користувачів на ролі для ефективної роботи системи, особистий кабінет для перегляду затраченого часу на виконані завдання, створення нових проектів та завдань, пошук по назві та сортування по даті, можливість обирати команду для виконання проекта, назначати відповідальну за завдання людину, ведення статистики затраченого часу на виконане завдання для подальшого аналізу, можливість назначити користувачу нову роль.

4. Покриття коду тестами було успішно виконане та пройдене. Тести забезпечують ефективну та якісну розробку продукту. При проведенні димового тестування були прийняті до уваги побажання, щодо покращення користувацького інтерфейсу.

5. Підведено підсумок переваг даного веб-застосунку перед аналогами, які присутні на ринку. Запропоновано ідеї для подальшого розширення та розвитку, визначено сфери застосування даної системи.

Виконання даної дипломної роботи, забезпечило набуття практичних навичок з технологіями для створення веб-застосунків. Створена система допоможе підвищити ефетивність управління проектами за допомогою чітко визначеної бізнес-логіки та надасть кожному розробнику вести свою статистику затраченого часу на виконання завдань.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Jira. Програмне забезпечення для відстеження завдань [Електронний ресурс]. Режим доступу: https://www.atlassian.com/ru/software/jira (дата звернення 17.04.2020)

2. Jira: подробное руководство для новичков [Електронний ресурс]. Режим доступу: https://bytextest.ru/2018/08/31/jira-dlya-novichkov/ (дата звернення 17.04.2020)

3. Що таке atlassian Jira? Переваги і недоліки [Електронний ресурс]. Режим доступу: https://softlist.com.ua/articles/chto-takoe-jira/ (дата звернення 18.04.2020)

4. Trello – універсальний менеджер проектів [Електронний ресурс]. Режим доступу: https://aweb.ua/blog/k-doske-trello-universalnyj-menedzher-proektov/ (дата звернення 18.04.2020)

5. Trello: що це таке і як користуватися [Електронний ресурс]. Режим доступу: https://blog.calltouch.ru/kak-polzovatsya-trello-instruktsiya-i-poleznye-hitrosti/ (дата звернення 19.04.2020)

6. Trello – что это, как пользоваться трелло эффективно [Електронний ресурс]. Режим доступу: https://www.houzz.ru/statyi/produktivnosty-trello-plaginy-dlya-domashnih-del-stsetivw-vs~98438061 (дата звернення 19.04.2020)

7. Any.DO – огляд сервісу [Електронний ресурс]. Режим доступу: https://startpack.ru/application/any-do-list (дата звернення 20.04.2020)

8. Список дел Напоминания Календарь Ежедневник | Any.do [Електронний ресурс]. Режим доступу: https://www.any.do/ru/ (дата звернення 21.04.2020)

9. TickTick – совершенный планировщик задач [Електронний ресурс]. Режим доступу: https://lifehacker.ru/ticktick/ (дата звернення 21.04.2020)

10. TickTick – 32 секретних фактів, огляд, характеристики [Електронний ресурс]. Режим доступу: https://rankquality.com/ticktick/ (дата звернення 21.04.2020)

11. TickTick [Електронний ресурс]. Режим доступу: https://ticktick.com/ (дата звернення 22.04.2020)

12. Клиент-серверная архитектура в картинках [Електронний ресурс]. Режим доступу: https://habr.com/ru/post/495698/ (дата звернення 23.04.2020)
13. Node Hero: Частина 1 - Починаємо роботу з Node.js [Електронний ресурс]. Режим доступу: https://codeguida.com/post/517 (дата звернення 23.04.2020)
14. Node.js V8 internals [Електронний ресурс]. Режим доступу: https://codeburst.io/node-js-v8-internals-an-illustrative-primer-83766e983bf6 (дата звернення 25.04.2020)
15. Java 8. Руководство для начинающих. Герберт Шилдт [Електронний ресурс]. Режим доступу: https://ru.bookmate.com/books/YZFuDJnQ (дата звернення 30.04.2020)
16. Spring Framework [Електронний ресурс]. Режим доступу: https://spring.io/projects/spring-framework (дата звернення 02.05.2020)
17. Веб-фреймворк Django (Python) - Изучение веб-разработки [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/ru/docs/Learn/Server-side/Django (дата звернення 05.05.2020)
18. Вивчення React. Повне керівництво по React [Електронний ресурс]. Режим доступу: https://learn-reactjs.ru/home (дата звернення 06.05.2020)
19. Руководство по Vue.js - METANIT.COM [Електронний ресурс]. Режим доступу: https://metanit.com/web/vuejs/ (дата звернення 09.05.2020)
20. AngularJS [Електронний ресурс]. Режим доступу: http://angular-doc.herokuapp.com/guide/overview (дата звернення 10.05.2020)
21. Документация к PostgreSQL 9.6.18 - Postgres Professional [Електронний ресурс]. Режим доступу: https://postgrespro.ru/docs/postgresql/9.6/index (дата звернення 10.05.2020)
22. What is HBase? | IBM [Електронний ресурс]. Режим доступу: https://www.ibm.com/analytics/hadoop/hbase (дата звернення 11.05.2020)
23. Паттерны для новичков: MVC vs MVP vs MVVM [Електронний ресурс]. Режим доступу: https://habr.com/ru/post/215605/ (дата звернення 11.05.2020)

ДОДАТОК 1

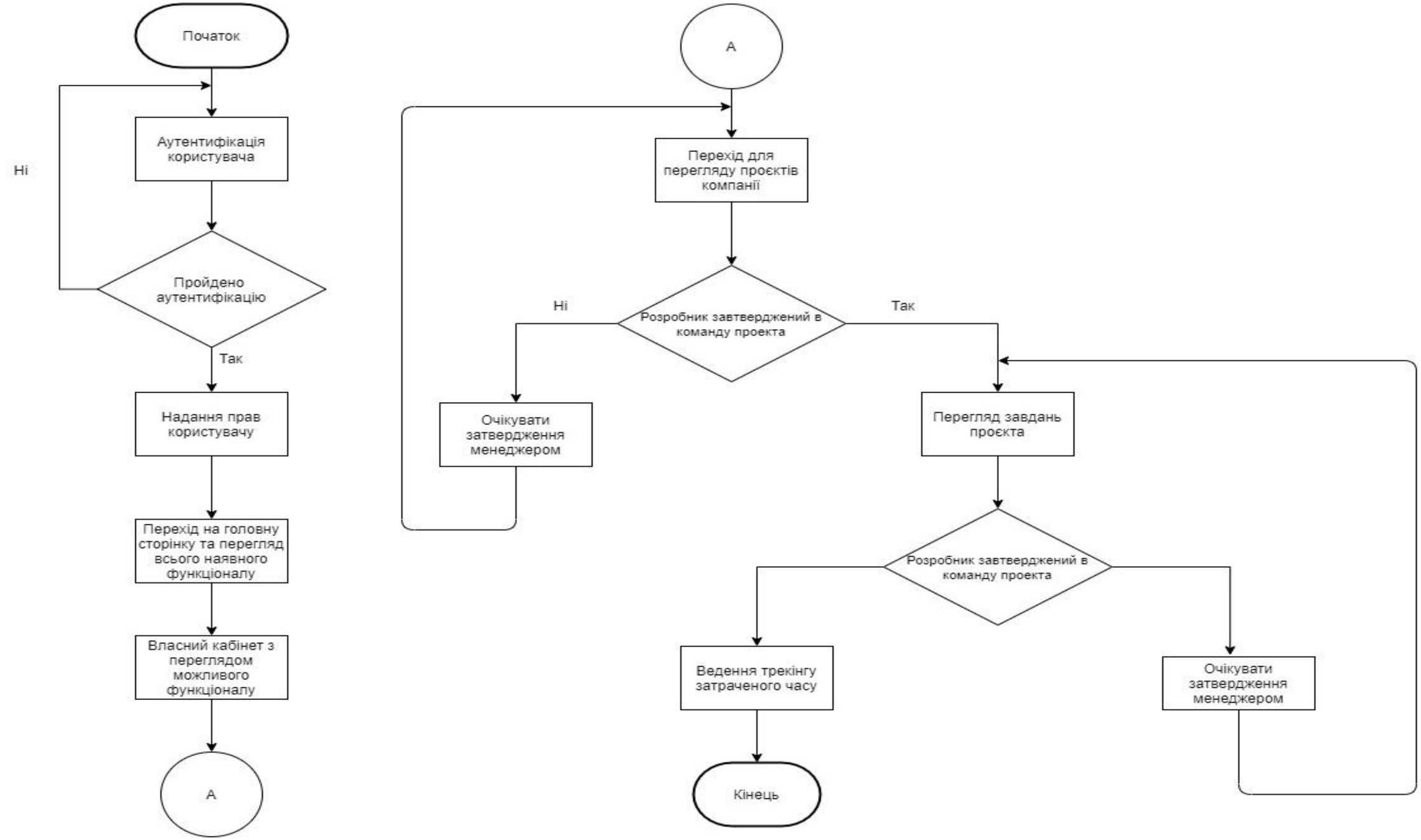
Веб-додаток інформаційної системи тайм-трекінгу

Алгоритм користувацьких дій

ІАЛЦ.467100.004.Д1

Аркушів 1

Київ – 2020 року



Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467100.004 Д1			
Розроб.		Царук В.В.			Веб додаток інформаційної системи тайм-трекінгу Алгоритм користувацьких дій	Літ.	Маса	Масштаб
Перевір.		Аленін О.І.						
						Арк.	Аркушів	
Н. контр.		Сімоненко В.П.			Дипломна робота	КПІ ФІОТ кафедра ОТ гр. ІП-64		
Затверд.								

ДОДАТОК 2

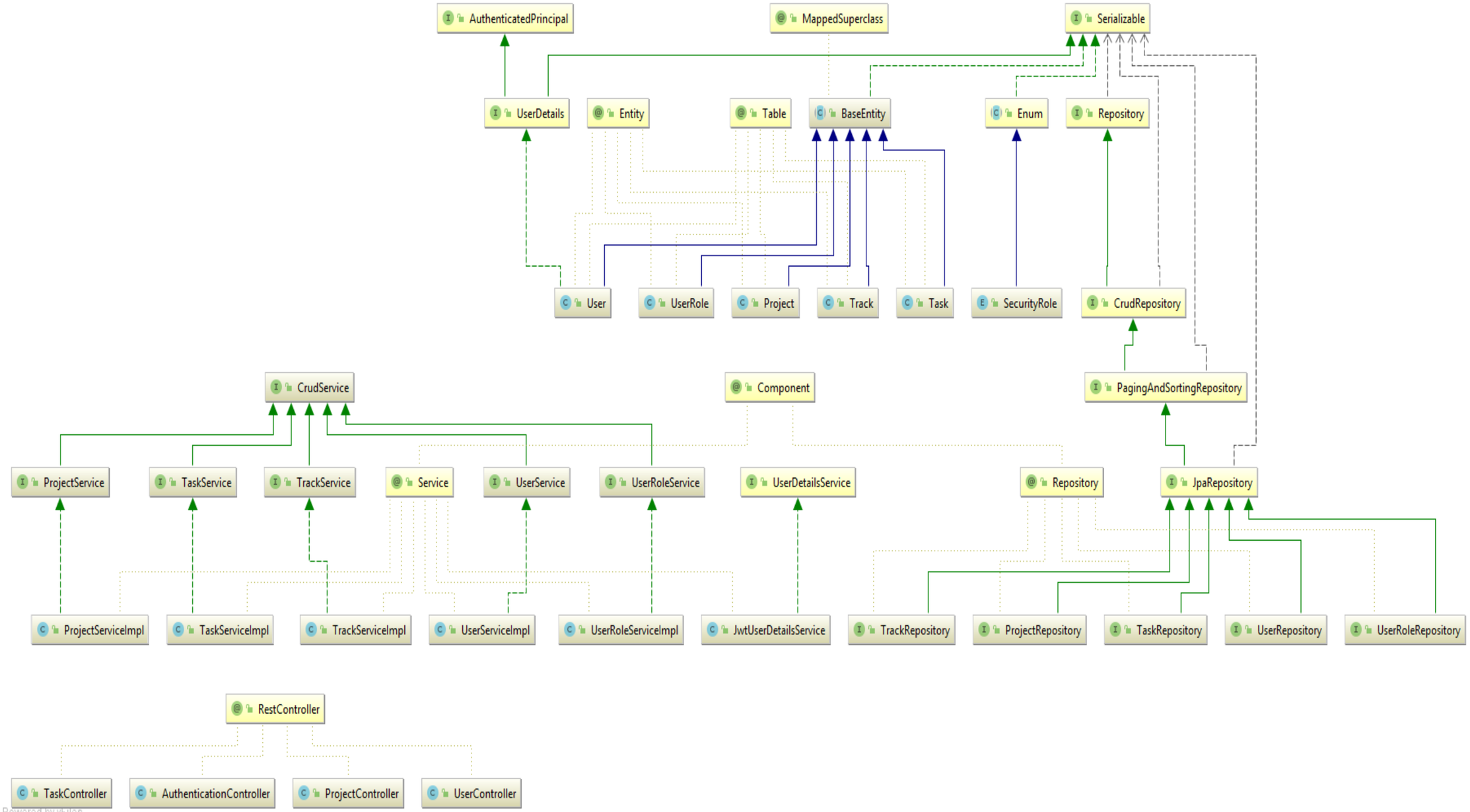
Веб-додаток інформаційної системи тайм-трекінгу

Функціональна схема

ІАЛЦ.467100.005 Д2

Аркушів 1

Київ – 2020 року



					ІАЛЦ.467100.005 Д2		
Зм.	Арк.	№ докум.	Підпис	Дата	*Веб додаток інформаційної системи тайм-трекінгу Функціональна схема*	Літ.	Маса / Масштаб
Розроб.		Царук В.В.					
Перевір.		Аленін О.І.				Арк.	Аркушів
Н. контр.		Сімоненко В.П.			*Дипломна робота*	*КПІ ФІОТ кафедра ОТ гр. ІП-64*	
Затверд.							

ДОДАТОК З

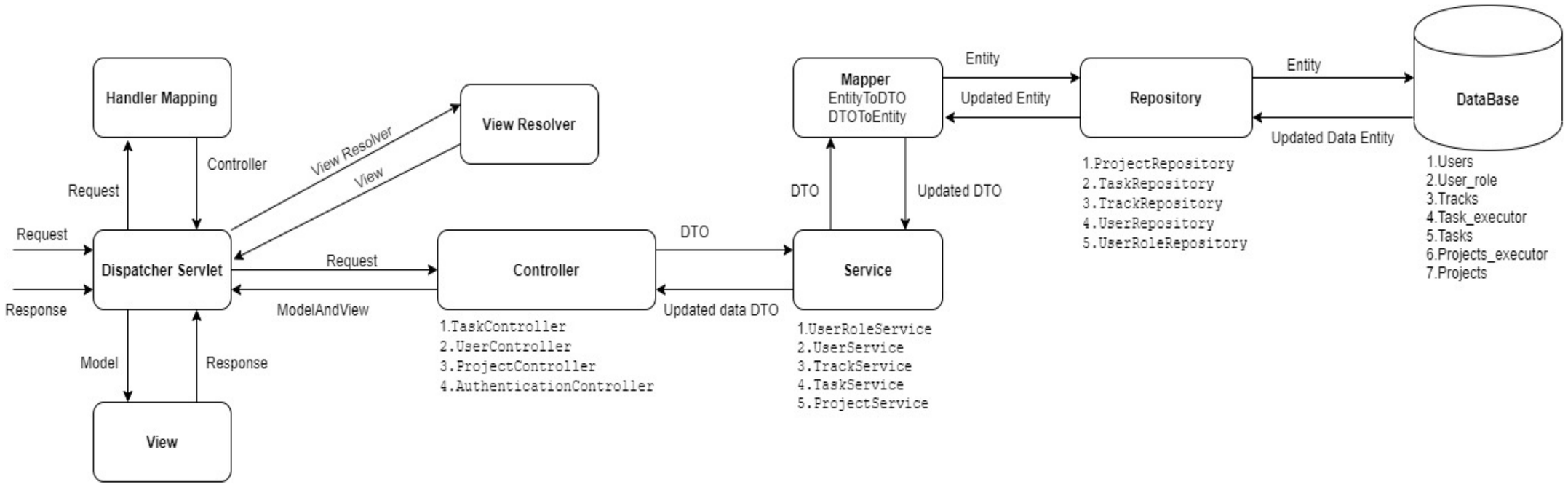
Веб-додаток інформаційної системи тайм-трекінгу

Структурна схема

ІАЛЦ.467100.006 Д3

Аркушів 1

Київ – 2020 р



					ІАЛЦ.467100.006 ДЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	*Веб додаток інформаційної системи тайм-трекінгу Структурна схема*	Літ.	Маса	Масштаб
Розроб.		Царук В.В.						
Перевір.		Аленін О.І.				Арк.	Аркушів	
Н. контр.		Сімоненко В.П.			*Дипломна робота*	*КПІ ФІОТ кафедра ОТ гр. ІП-64*		
Затверд.								

ДОДАТОК 4

Веб-додаток інформаційної системи тайм-трекінгу

Текст програми

ІАЛЦ.467100.007 Д4

Аркушів 10

Київ – 2020 року

```java
@MappedSuperclass
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@EntityListeners(BaseEntityListener.class)
public abstract class BaseEntity implements Serializable{

    private static final long serialVersionUID = 1L;

    private Long id;

    private Date creatingDate;

    private Date editingDate;

    public void update(BaseEntity entity){
        if (entity != null) {
            setId(entity.getId());
        }
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID", unique = true, nullable = false)
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @JsonFormat(pattern="HH:mm dd.MM.yy")
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "CREATING_DATE", unique = false, nullable = true)
    public Date getCreatingDate() {
        return creatingDate;
    }

    public void setCreatingDate(Date creatingDate) {
        this.creatingDate = creatingDate;
    }

    @JsonFormat(pattern="HH:mm dd.MM.yy")
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "EDITING_DATE", unique = false, nullable = true)
    public Date getEditingDate() {
        return editingDate;
    }

    public void setEditingDate(Date editingDate) {
        this.editingDate = editingDate;
    }
}
@Entity
@Table(name="PROJECTS")
@AttributeOverride(name = "id", column = @Column(name = "PROJECT_ID",
unique=true, nullable=false))
public class Project extends BaseEntity {

    private String name;

    @JsonIgnore
```

```
    private User owner;


    @JsonIgnore
    private Set<User> executors = new HashSet<>(0);

    @JsonIgnore
    private Set<Task> tasks = new HashSet<>(0);

    public Project(){}

    public Project(String name, User owner){
        setName(name);
        setOwner(owner);
    }

    public void update(Project project) {
        super.update(project);
        if (project != null){
            setName(project.getName());
            setOwner(project.getOwner());
            setExecutors(project.getExecutors());
            setTasks(project.getTasks());
        }
    }
    @Column(name = "PROJECT_NAME", unique = false, nullable = false)
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "USER_OWNER_ID", nullable = false)
    public User getOwner() {
        return owner;
    }

    public void setOwner(User owner) {
        this.owner = owner;
    }

    @Transient
    public Set<User> getExecutors() {
        return executors;
    }

    public void setExecutors(Set<User> executors) {
        this.executors = executors;
    }

    @OneToMany(fetch = FetchType.LAZY, mappedBy = "project")
    public Set<Task> getTasks() {
        return tasks;
    }

    public void setTasks(Set<Task> tasks) {
        this.tasks = tasks;
    }

    @Override
```

```
    public String toString() {
        return "Project [id=" + getId() +
                ", name=" + getName() +
                "]";
    }

@Entity
@Table(name="TASKS")
@AttributeOverride(name = "id", column = @Column(name = "TASK_ID", unique=true,
nullable=false))
public class Task extends BaseEntity{

    private String name;

    @JsonIgnore
    private User owner;

    private Project project;

    @JsonIgnore
    private Set<User> executors = new HashSet<>(0);

    public Task(){}

    public Task(String name, User owner, Project project){
        setName(name);
        setOwner(owner);
        setProject(project);
    }

    public void update(Task task) {
        super.update(task);
        if (task != null){
            setName(task.getName());
            setOwner(task.getOwner());
            setProject(task.getProject());
            setExecutors(task.getExecutors());
        }
    }

    @Column(name = "TASK_NAME", unique = false, nullable = false)
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "USER_OWNER_ID", nullable = false)
    public User getOwner() {
        return owner;
    }

    public void setOwner(User owner) {
        this.owner = owner;
    }

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "PROJECT_ID", nullable = false)
    public Project getProject() {
```

```
        return project;
    }

    public void setProject(Project project) {
        this.project = project;
    }

    @Transient
    public Set<User> getExecutors() {
        return executors;
    }

    public void setExecutors(Set<User> executors) {
        this.executors = executors;
    }

    @Override
    public String toString() {
        return "Task [id=" + getId() +
                ", name=" + getName() +
                "]";
    }
}
@Entity
@Table(name="TRACKS")
@AttributeOverride(name = "id", column = @Column(name = "TRACK_ID", unique=true,
nullable=false))
public class Track extends BaseEntity {

    private String name;

    private User owner;

    private Task task;

    private Long time;

    private Date dateActivity;

    public Track(){}

    public Track(String name, User owner, Task task, Long time){
        setName(name);
        setOwner(owner);
        setTask(task);
        setTime(time);
    }

    public void update(Track track) {
        super.update(track);
        if (track != null){
            setName(track.getName());
            setOwner(track.getOwner());
            setTask(track.getTask());
            setTime(track.getTime());
            setDateActivity(track.getDateActivity());
        }
    }

    @Column(name = "TRACK_NAME", unique = false, nullable = false)
    public String getName() {
        return name;
```

```
    }

    public void setName(String name) {
        this.name = name;
    }

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "USER_OWNER_ID", nullable = false)
    public User getOwner() {
        return owner;
    }

    public void setOwner(User owner) {
        this.owner = owner;
    }

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "TASK_ID", nullable = false)
    public Task getTask() {
        return task;
    }

    public void setTask(Task task) {
        this.task = task;
    }

    @Column(name = "TIME_VALUE", unique = false, nullable = false)
    public Long getTime() {
        return time;
    }

    public void setTime(Long time) {
        this.time = time;
    }

    @JsonFormat(pattern="HH:mm dd.MM.yy")
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "DATE_ACTIVITY", unique = false, nullable = false)
    public Date getDateActivity() {
        return dateActivity;
    }

    public void setDateActivity(Date dateActivity) {
        this.dateActivity = dateActivity;
    }

    @Override
    public String toString() {
        return "Track [id=" + getId() +
                ", name=" + getName() +
                "]";
    }
}
@Entity
@Table(name="USERS")
@AttributeOverride(name = "id", column = @Column(name = "USER_ID", unique=true,
nullable=false))
public class User extends BaseEntity implements UserDetails{

    private String username;

    private String password;
```

```
    private UserRole userRole;

    private String firstName;

    private String lastName;

    private String email;

    @JsonIgnore
    private Set<UserAuthority> authorities = new HashSet<>(0);

    private boolean accountNonExpired = true;

    private boolean accountNonLocked = true;

    private boolean credentialsNonExpired = true;

    private boolean enabled = true;

    private Date latestActivity;

    @JsonIgnore
    private Set<Comment> comments = new HashSet<>(0);

    @JsonIgnore
    private Set<Project> projects = new HashSet<>(0);

    public User(){}

    public User(String userName, String password){
        setUsername(userName);
        setPassword(password);
    }

    public User(String userName, String password, UserRole userRole, String
firstName, String lastName, String email){
        setUsername(userName);
        setPassword(password);
        setUserRole(userRole);
        setFirstName(firstName);
        setLastName(lastName);
        setEmail(email);
    }

    public void update(User user) {
        super.update(user);
        if (user != null){
            setUsername(user.getUsername());
            setPassword(user.getPassword());
            setUserRole(user.getUserRole());
            setFirstName(user.getFirstName());
            setLastName(user.getLastName());
            setEmail(user.getEmail());
            setLatestActivity(user.getLatestActivity());
        }
    }

    @Column(name="FIRST_NAME", unique=false, nullable=false)
    public String getFirstName() {
        return firstName;
    }
```

```
public void setFirstName(String firstName) {
    this.firstName = firstName;
}

@Column(name="LAST_NAME", unique=false, nullable=false)
public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

@Column(name="USER_EMAIL", unique=true, nullable=false)
public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

@OneToMany(fetch = FetchType.LAZY, mappedBy = "user")
public Set<Comment> getComments() {
    return comments;
}

public void setProjects(Set<Project> projects) {
    this.projects = projects;
}

@OneToMany(fetch = FetchType.LAZY, mappedBy = "owner")
public Set<Project> getProjects() {
    return projects;
}

public void setComments(Set<Comment> comments) {
    this.comments = comments;
}

@Column(name="USER_NAME", unique=true, nullable=true)
public String getUsername() {
    return username;
}

public void setUsername(String userName) {
    this.username = userName;
}

@JsonIgnore
@Column(name="USER_PASSWORD", unique=false, nullable=false)
@JsonProperty(value = "password")
public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

@Transient
```

```
    @Override
    public Set<UserAuthority> getAuthorities() {
        return authorities;
    }

    @JsonIgnore
    @Column(name="ACCOUNT_NON_EXPIRED", unique=false, nullable=false)
    @Override
    public boolean isAccountNonExpired() {
        return accountNonExpired;
    }

    public void setAccountNonExpired(boolean accountNonExpired) {
        this.accountNonExpired = accountNonExpired;
    }

    @JsonIgnore
    @Column(name="ACCOUNT_NON_LOCKED", unique=false, nullable=false)
    @Override
    public boolean isAccountNonLocked() {
        return accountNonLocked;
    }

    public void setAccountNonLocked(boolean accountNonLocked) {
        this.accountNonLocked = accountNonLocked;
    }

    @JsonIgnore
    @Column(name="CREDENTIALS_NON_EXPIRED", unique=false, nullable=false)
    @Override
    public boolean isCredentialsNonExpired() {
        return credentialsNonExpired;
    }

    public void setCredentialsNonExpired(boolean credentialsNonExpired) {
        this.credentialsNonExpired = credentialsNonExpired;
    }

    @JsonIgnore
    @Column(name="ACCOUNT_ENABLED", unique=false, nullable=false)
    public boolean isEnabled() {
        return enabled;
    }

    public void setEnabled(boolean enabled) {
        this.enabled = enabled;
    }

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "USER_ROLE_ID", nullable = true)
    public UserRole getUserRole() {
        return userRole;
    }

    public void setUserRole(UserRole userRole) {
        this.userRole = userRole;
        if (userRole != null && userRole.getSecurityRole() != null) {
            this.authorities.add(new
UserAuthority(userRole.getSecurityRole().name()));
        }
    }
```

```
    @JsonIgnore
    @Override
    public Date getEditingDate() {
        return super.getEditingDate();
    }

    @JsonFormat(pattern="HH:mm dd.MM.yy")
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "LATEST_ACTIVITY_DATE", unique = false, nullable = true)
    public Date getLatestActivity() {
        return latestActivity;
    }

    public void setLatestActivity(Date latestActivity) {
        this.latestActivity = latestActivity;
    }

    @Override
    public String toString() {
        return "User [id=" + getId() +
                ", userName=" + getUsername() +
                ", firstName=" + getFirstName() +
                ", lastName=" + getLastName()+
                "]";
    }
}
# Connection settings
hibernate.connection.driver_class=org.postgresql.Driver
hibernate.connection.url=jdbc:postgresql://localhost:5432/timetrackingsystem
hibernate.default_schema=timetrackingschema
hibernate.connection.username=postgres
hibernate.connection.password=root

# SQL settings
sql.initializer.ddlmode=update
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

# Additional properties
hibernate.bytecode.use_reflection_optimizer=false
hibernate.show_sql=false

@Repository
public interface ProjectRepository extends JpaRepository<Project, Long> {
}
@Repository
public interface TaskRepository extends JpaRepository<Task, Long> {
}

<div style="display: flex; flex-direction: column; align-items: stretch; height:
100%;">
  <form class="register-card"
        (ngSubmit)="onSubmit()"
        [formGroup]="signInForm">
    <!-- TOOLBAR -->
    <mat-toolbar color="primary">
      <mat-toolbar-row>
        <mat-icon class="tool-bar-item">person_pin</mat-icon>
        <span class="tool-bar-item"
              >{{ 'signInForm.title' | translate }}</span>
      </mat-toolbar-row>
    </mat-toolbar>
    <!-- REGISTER CARD -->
```

```
    <mat-card>
      <!-- CONTENT -->
      <mat-card-content class="register-card-content" >
        <!-- USERNAME -->
        <mat-form-field >
          <input matInput
                 placeholder="{{ 'signInForm.username.placeholder' | translate
}}"
                 required
                 minlength="{{minLengthUserName}}"
                 maxlength="{{maxLengthUserName}}"
                 formControlName="userName"
                 [(ngModel)]="loginedUser.username"
                 [errorStateMatcher]="formControlMatcher">
          <mat-error *ngIf="isErrorActive(userNameFormControl, LENGTH_ERROR)"
                     [translate]="'signInForm.username.lengthHint'"
                     [translateParams]="{
                        'minLengthUserName': minLengthUserName,
                        'maxLengthUserName': maxLengthUserName
                     }"
                     >Length error</mat-error>
          <mat-error *ngIf="isErrorActive(userNameFormControl, REQUIRED_ERROR)"
                     [translate]="'validation.requiredField'"
                     >Required error</mat-error>
        </mat-form-field>
        <!-- USER PASSWORD -->
        <mat-form-field>
          <input matInput
                 placeholder="{{ 'signInForm.password.placeholder' | translate
}}"
                 required
                 minlength="{{minLengthPassword}}"
                 maxlength="{{maxLengthPassword}}"
                 formControlName="password"
                 [(ngModel)]="loginedUser.password"
                 [errorStateMatcher]="formControlMatcher"
                 [type]="hidePassword ? 'password' : 'text'">
          <mat-icon matSuffix
                    (click)="hidePassword = !hidePassword"
                    >{{hidePassword ? 'visibility' : 'visibility_off'}}</mat-
icon>
          <mat-error *ngIf="isErrorActive(userPasswordFormControl,
PASSWORD_ERROR)"
                     [translate]="'signInForm.password.patternHint'"
                     >Password error</mat-error>
          <mat-error *ngIf="isErrorActive(userPasswordFormControl,
LENGTH_ERROR)"
                     [translate]="'signInForm.password.lengthHint'"
                     [translateParams]="{
                        'minLengthPassword': minLengthPassword,
                        'maxLengthPassword': maxLengthPassword
                     }"
                     >Length error</mat-error>
          <mat-error *ngIf="isErrorActive(userPasswordFormControl,
REQUIRED_ERROR)"
                     [translate]="'validation.requiredField'"
                     >Required error</mat-error>
        </mat-form-field>
        <!-- PROGRESS -->
        <div *ngIf="isInProgress">
          <br/>
          <mat-progress-bar mode="indeterminate"></mat-progress-bar>
```